The University of Texas at Austin
Electrical and Computer Engineering
Cockrell School of Engineering

Spring 2023

ADVANCED TOPICS IN COMPUTER VISION

Atlas Wang
Assistant Professor, The University of Texas at Austin

Visual Informatics Group@UT Austin
https://vita-group.github.io/

Course Logistics

- We meet on MW 10:30am - noon (BUR 130)
  - Do I have to come to the classroom, or can I audit?
  - After-class communication: **Slack (link sent)** – **IMPORTANT!**
  - Class materials are distributed on **Course Webpage (NOT Canvas)**: https://vita-group.github.io/spring_23.html
- We do not follow any textbook closely. Instead, we have many “recommended materials”.
- **Instructor Office Hour:** 11am – noon every Tuesday, at EER office 6.886
- **TA Office Hour:** 4 - 5 pm every Thursday, at EER 3.854
- Online Q&A: ***anytime, just ask on Slack!***

Overview & Prerequisite

- Computer vision is a HUGE field. This class is designed to cover just “several drops” in the ocean, focusing on the “hot and fresh” frontiers (I have to re-make 40-50% slides since its last offering)
- Lectures are mixture of detailed techniques and high-level ideas.
- **This class is NOT designed for pure “beginners”**
  - We will speak technical language quickly from Day 1
  - You are assumed to already be familiar with: *Linear Algebra, Convex Optimization, Probability & Stochastic Process, Image & Video Processing*
  - You are assumed to know many basics about (but not an expert on): *Digital Signal Processing, Image & Video Processing, Machine Learning & Data Mining*

My Assumption (a.k.a. Your Expectation)

- You have some basic understanding of Deep Learning
  - e.g., you should have already heard about LeNet or AlexNet; known what convolution or fully-connected layers were; or tried things like MNIST classification before
- You are comfortable with Python & PyTorch, and enjoy ”keeping hands dirty”
  - Since this is a graduate-facing class, **NO** additional coding or basic data science “crush course” will be offered
- You are prepared to pay attention to our intensive, fast-forwarding contents, every 75-minute class
  - This field is developed at an unprecedentedly high pace, so will this class be

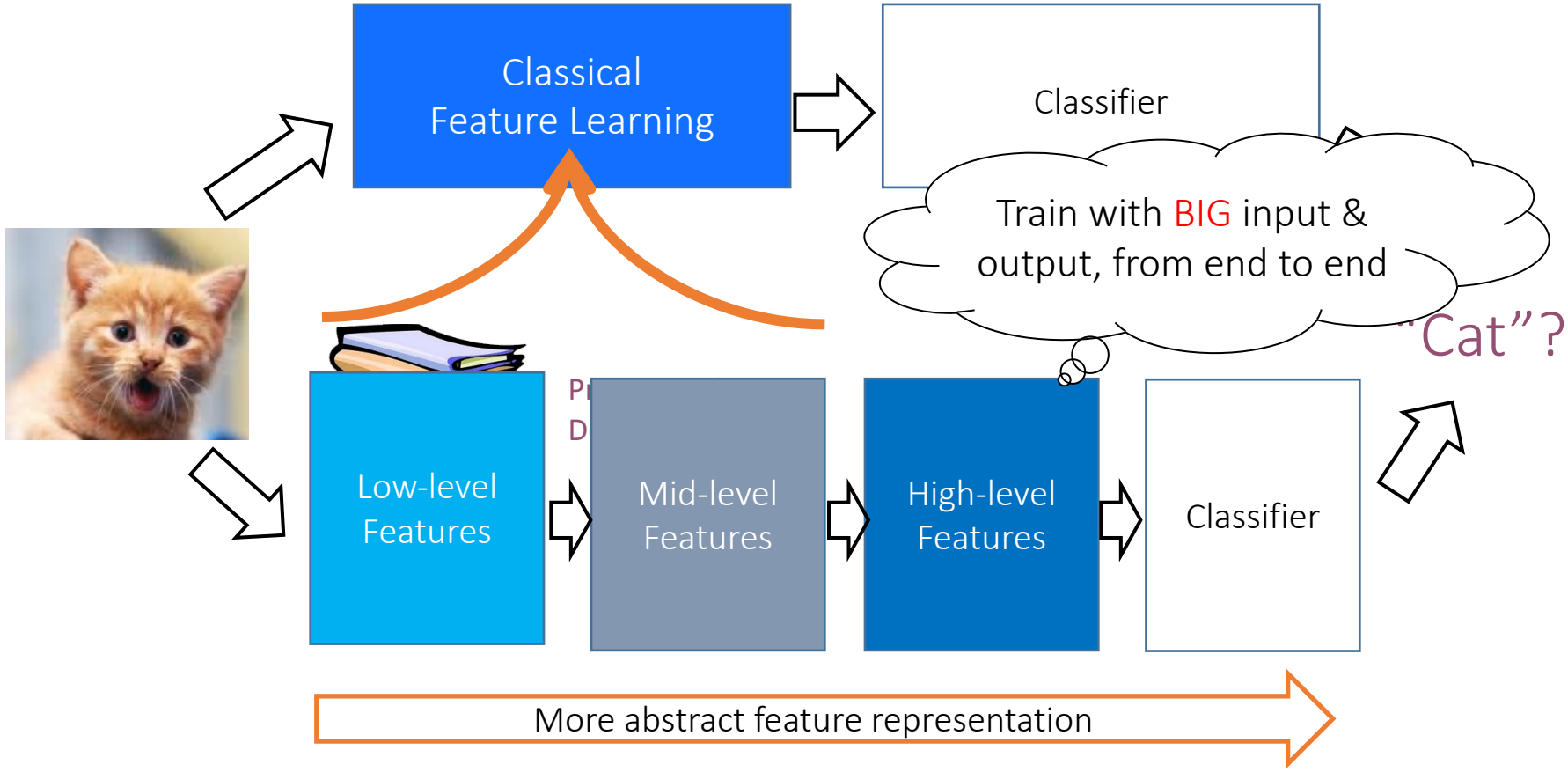
Grading Policy

- Class Participation: 10% (what does this mean?)
- Mid-term exam: 20% (Time TBD).
- Final Project: 70%
  - Progress report (**15%**) Due by the end of **Week 8 (3/05 Sunday)**: 2-Page report, including project title, team member, problem description, preliminary literature survey, the proposed technical plan, and references
  - Presentation (**20%**): Be prepared to be challenged by your peers and the instructor
  - Code review **(15%):** Write clean, well-documented and runnable codes, PLEASE
  - Final Report **(20%): (**8+1)-page report following the standard CVPR paper template (and quality level)
    - Template: http://cvpr2020.thecvf.com/sites/default/files/2019-09/cvpr2020AuthorKit.zip

Project Guidance

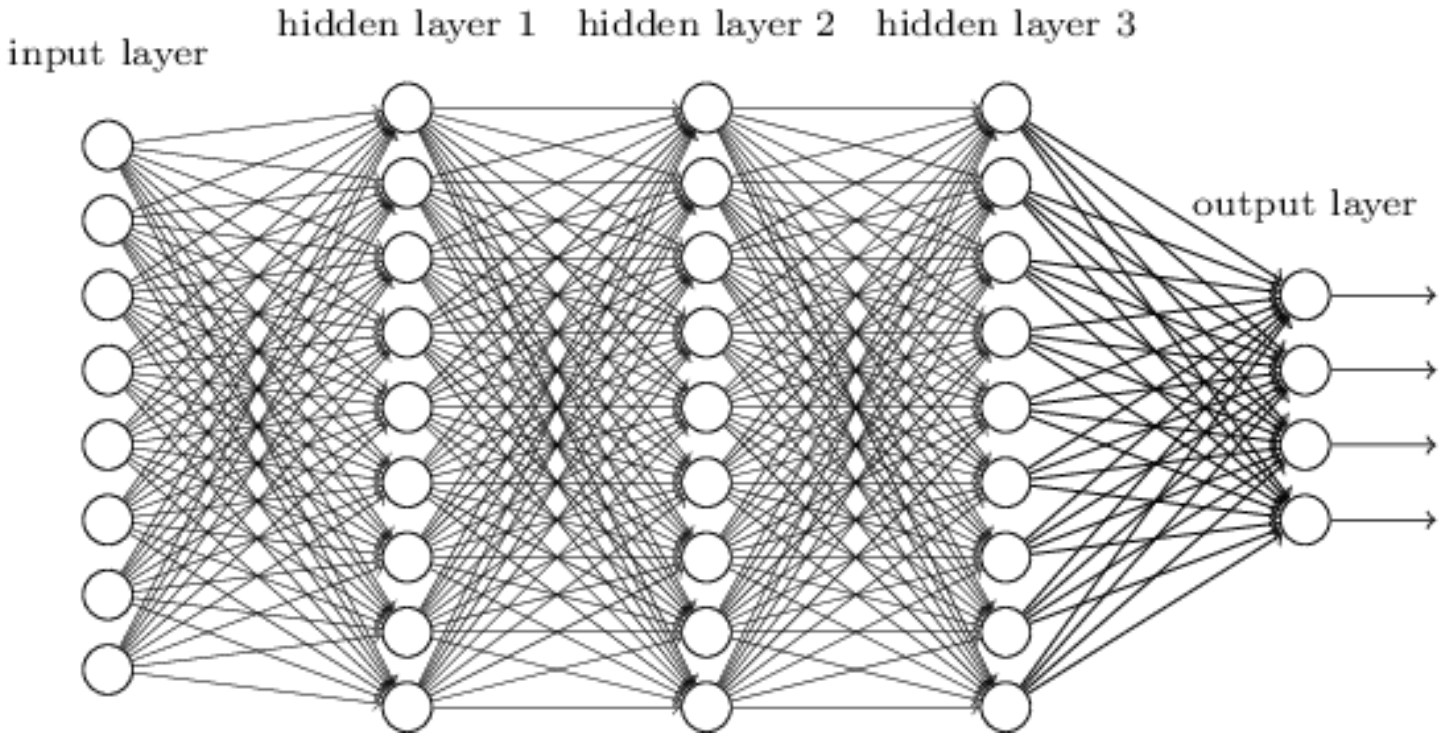
- **Teaming:** we encourage **2-3 students** to form a team, as you are expected to carry on a semester-long research project with substantial innovations.
    - Teams with more or fewer members may be well justified to be approved by the instructor
    - You are encouraged to use the slack channel *"project_team"* to recruit teammates
- Each project team must be registered to and approved by the end of **Week 7 (2/26 Sunday)**.
    - A Google Sheet will be provided for team registration
- **Topic:** your choice, but must be relevant to computer vision
    - What if I don't have a specific idea now ? Talk to the instructor & the TA …
- **Some good suggestions developed** by TAs before: https://vita-group.github.io/Fall22/0901_project.pdf
    - *How to develop a good project timeline? How to write good project proposal and report? Any example or "template"?*

Feature learning: Going Deep



Deep learning

- Learn a *feature hierarchy* all the way from raw inputs (e.g. pixels) to classifier
- Each layer extracts features from the output of previous layer
- Train all layers jointly



Status Quo

AlexNet, 8 layers
(ILSVRC 2012)

VGG, 19 layers
(ILSVRC 2014)

ResNet, 152 layers
(ILSVRC 2015)

Current Trend:

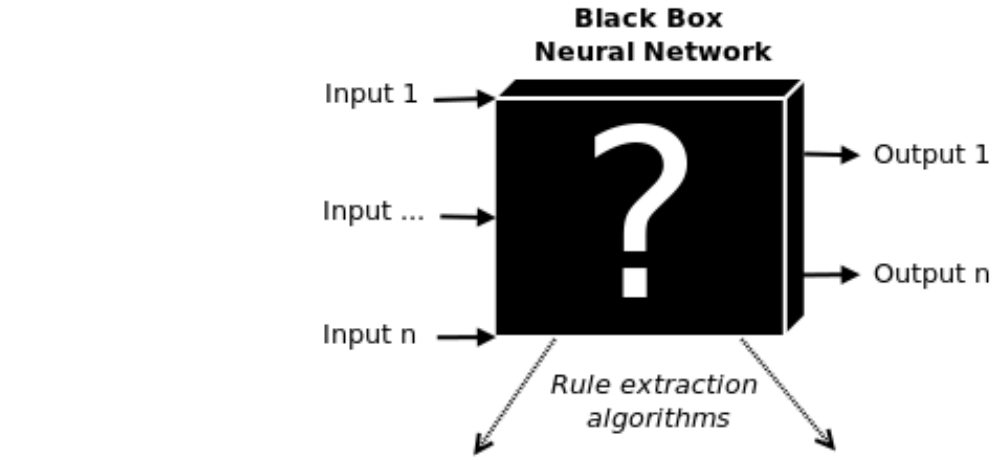
- To build increasingly larger, deeper networks, trained with more massive data, based on the benefits of high-performance computing.
- Play with the connectivity and add “skips”

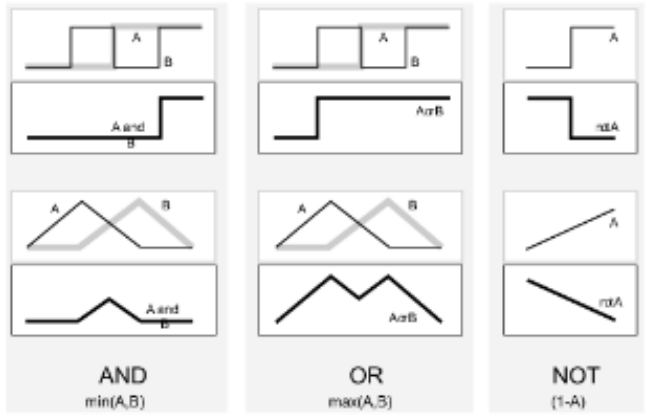

I WAS WINNING IMAGENET
UNTIL A DEEPER MODEL CAME ALONG
imgflip.com


专业深度学习调试
caffe安装 10元
CNN 5元/层
RNN 8元/层
专业

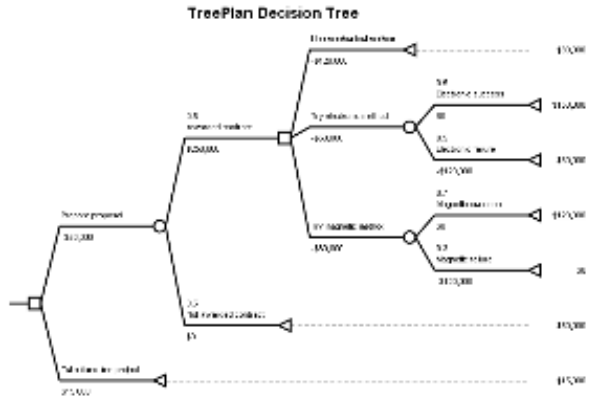
Grand Challenges

- Why/how deep learning works?
  - *In theory, many cases shouldn't even work…*
  - Gap between engineering (or art) and science: Lack of theoretical understandings & guarantees, and analytical tools
  - Training is computationally expensive and difficult, relying on many "magics"
  - No principled way to incorporate domain expertise, or to interpret the model behaviors



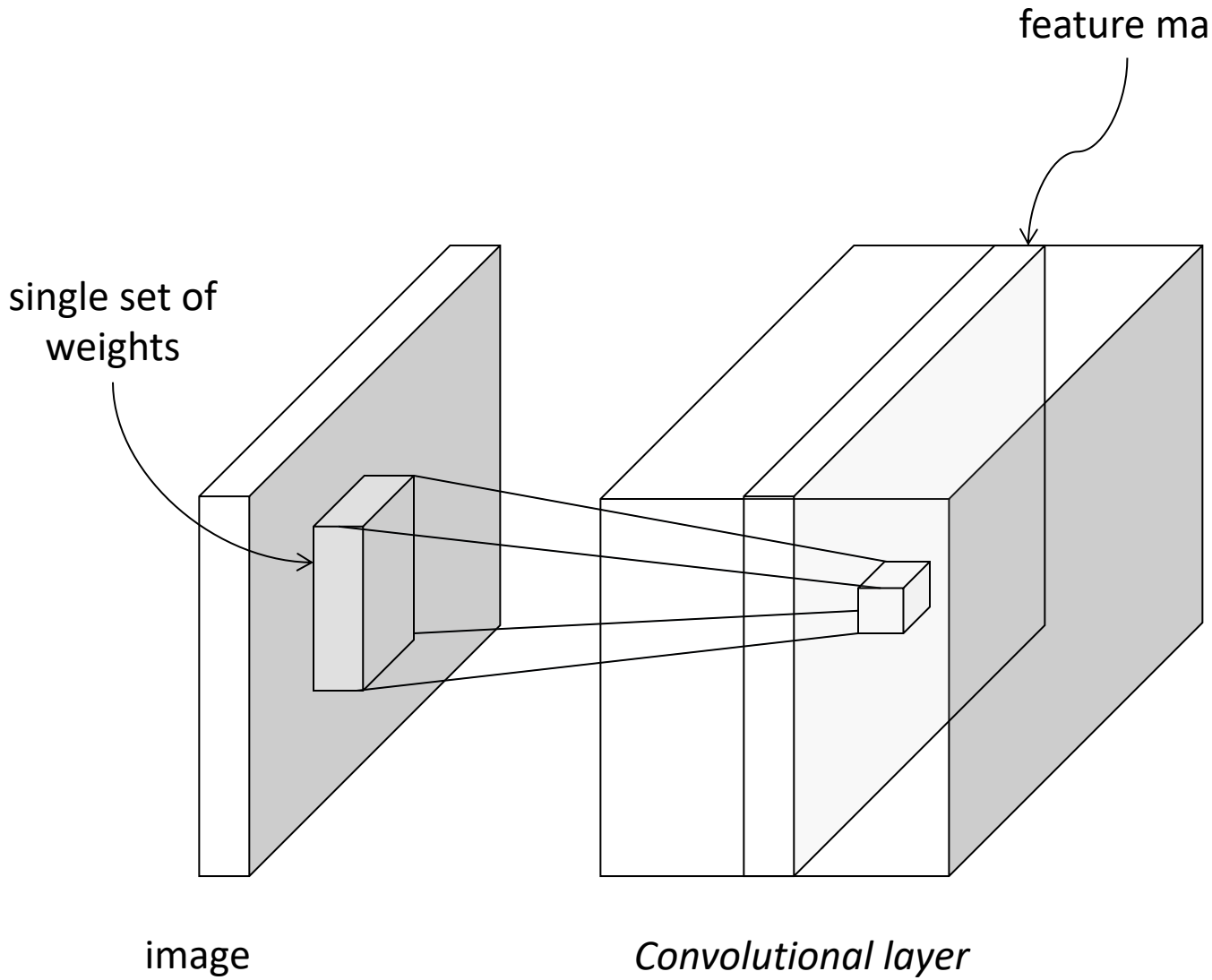


Extract Logic in either symbolic or fuzzy form

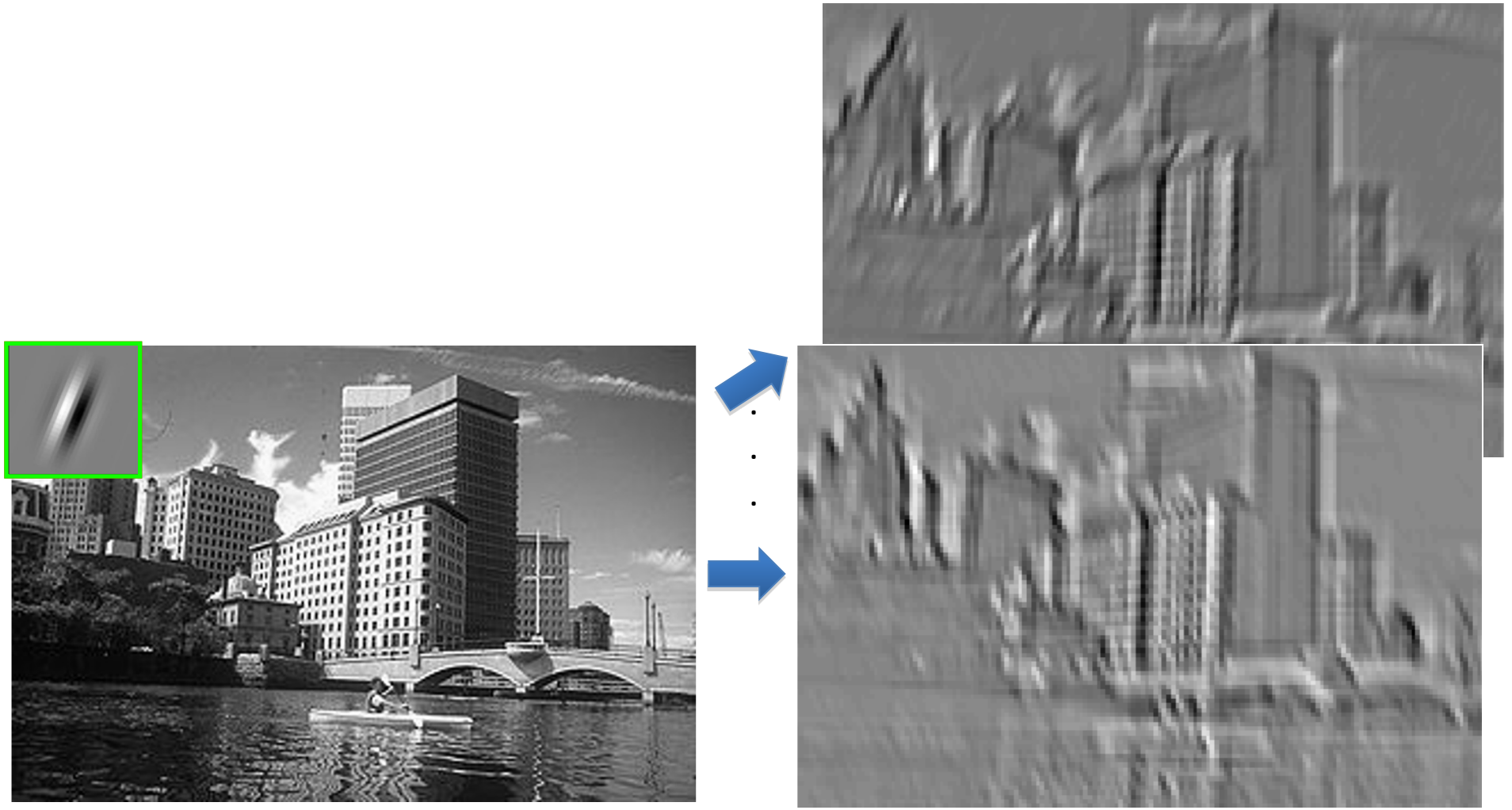


Extract Decision Trees contraining Logic

From fully connected to convolutional networks



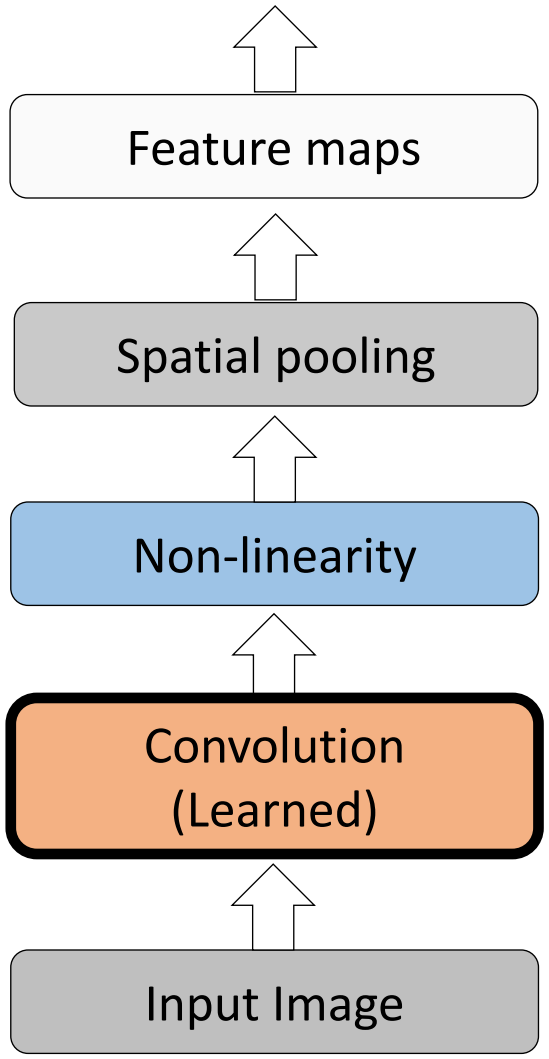
Convolution as feature extraction

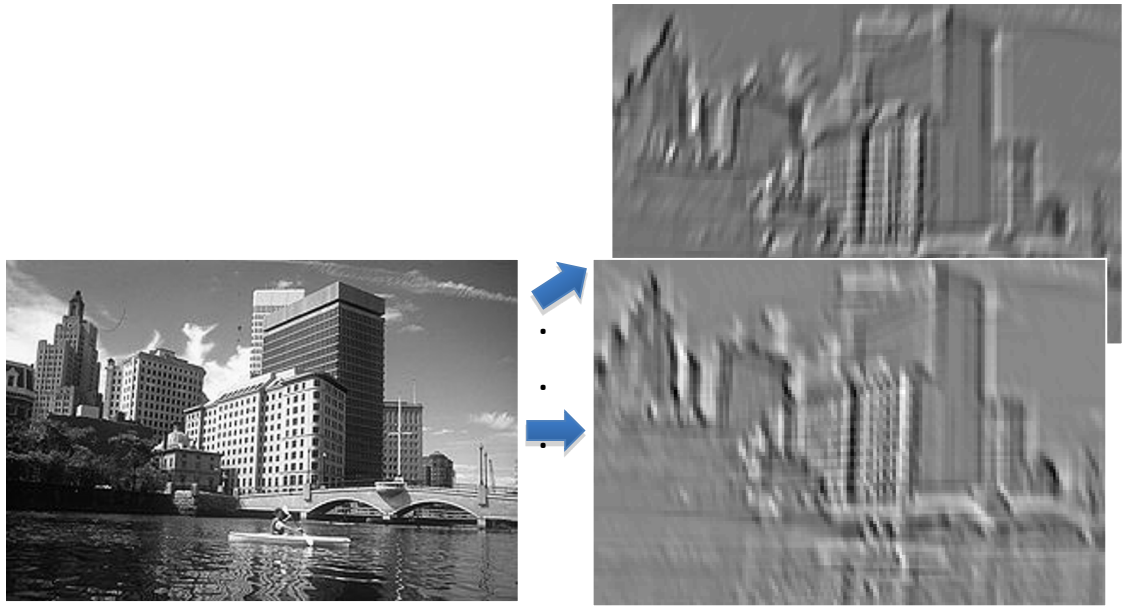


Input

Feature Map

Key operations in a CNN



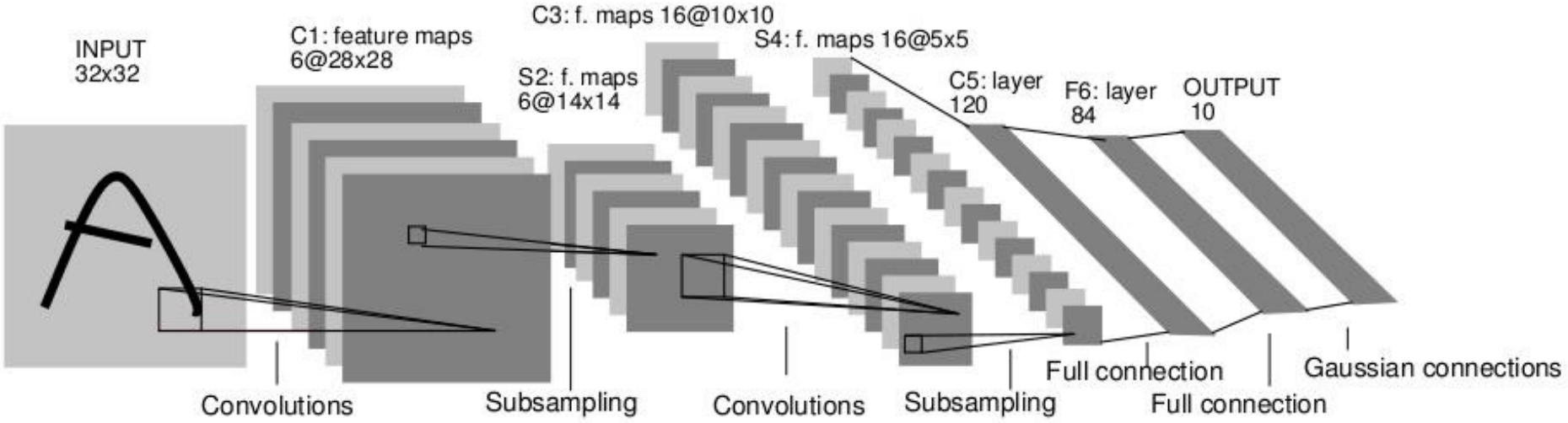


Input

Feature Map

Source: R. Fergus, Y. LeCun

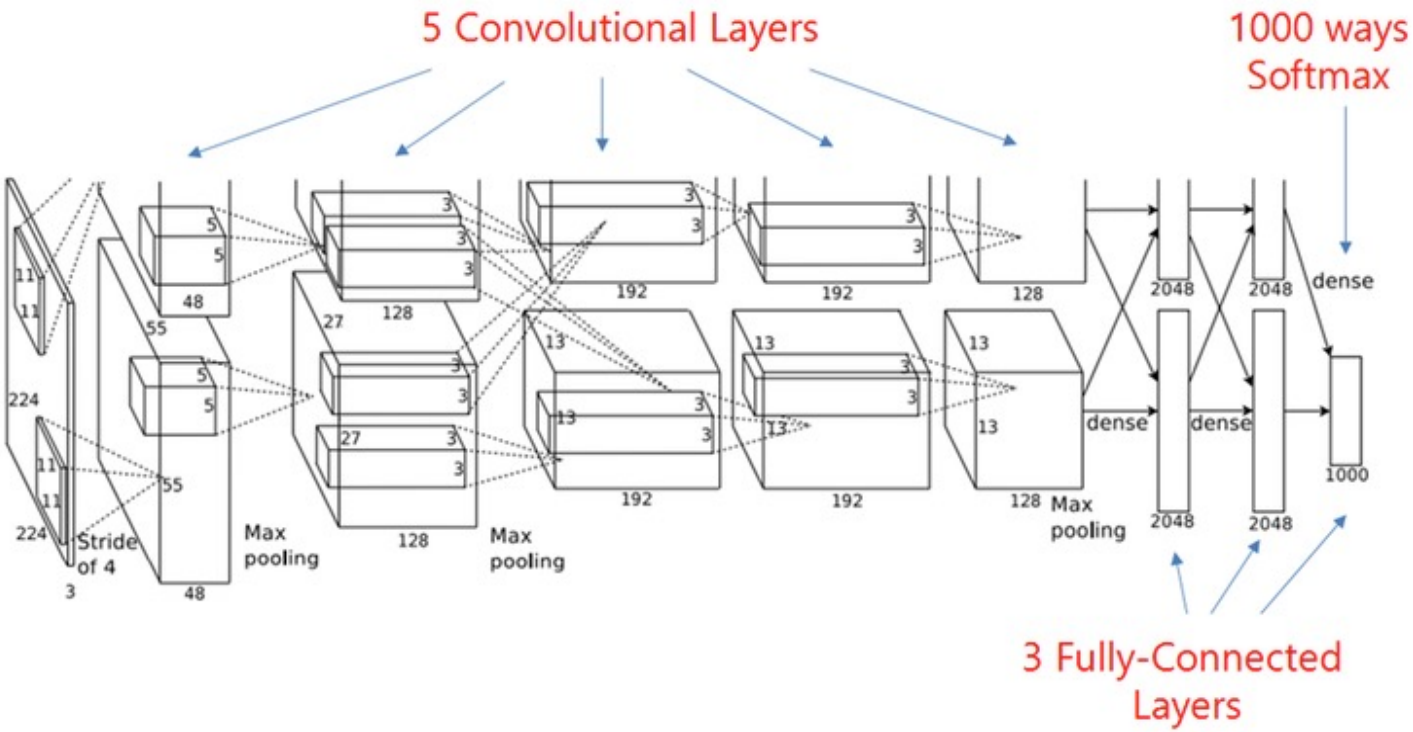
LeNet-5



- Average pooling
- Sigmoid or tanh nonlinearity
- Fully connected layers at the end
- Trained on MNIST digit dataset with 60K training examples

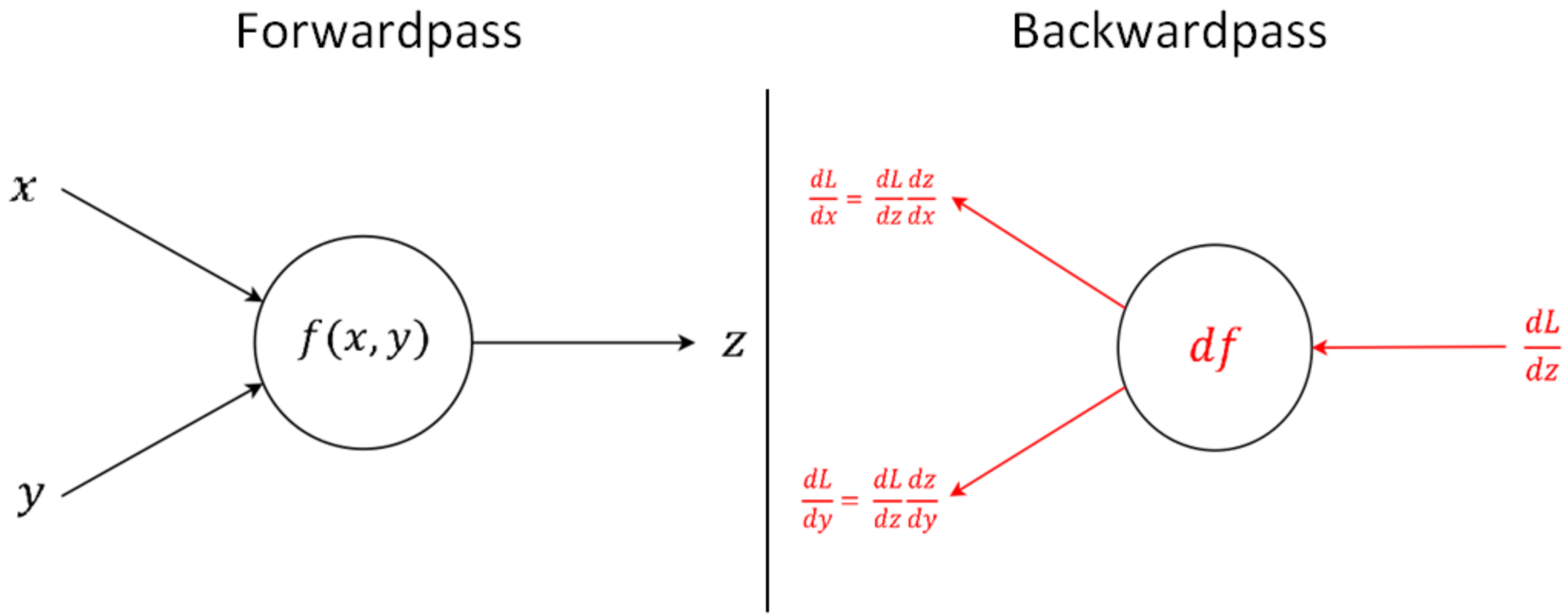
Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86(11): 2278–2324, 1998.

AlexNet, 2012

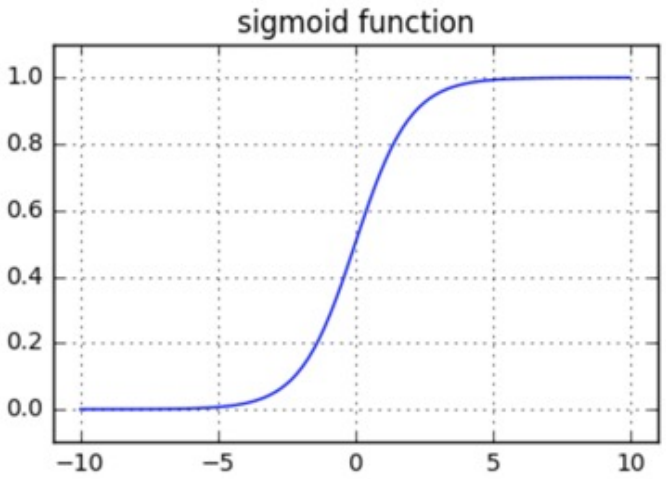


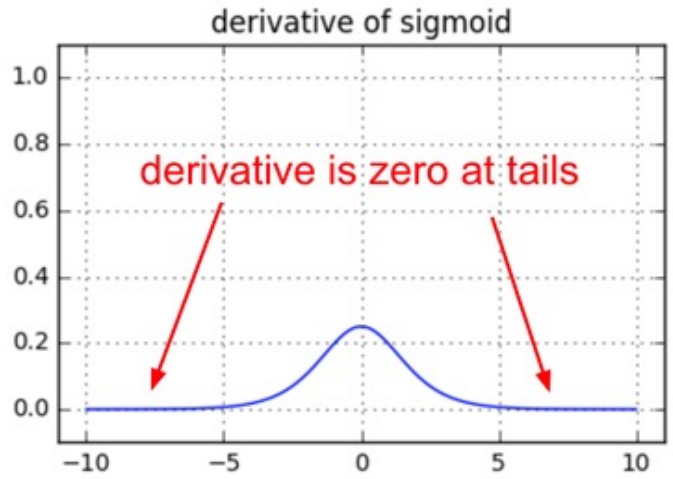
- The **FIRST** winner deep model in computer vision, and one of the most classical choices for domain experts to adapt for their applications
- 5 convolutional layers + 3 fully-connected layers + softmax classifier
- Three Key Design Features: **ReLU, dropout, data augmentation**

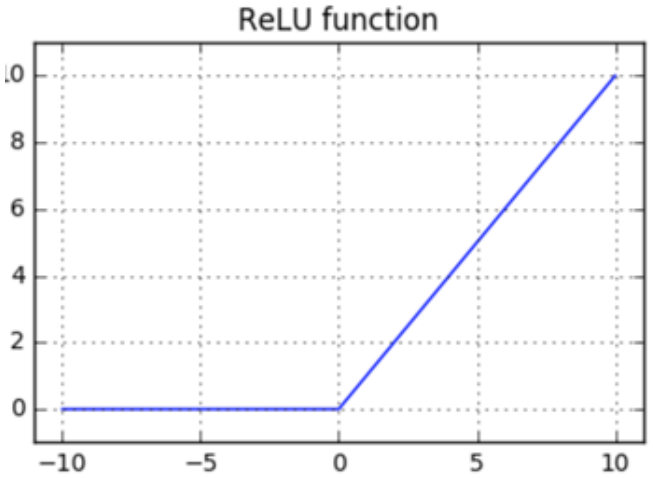
Recap: “Chain Rule”

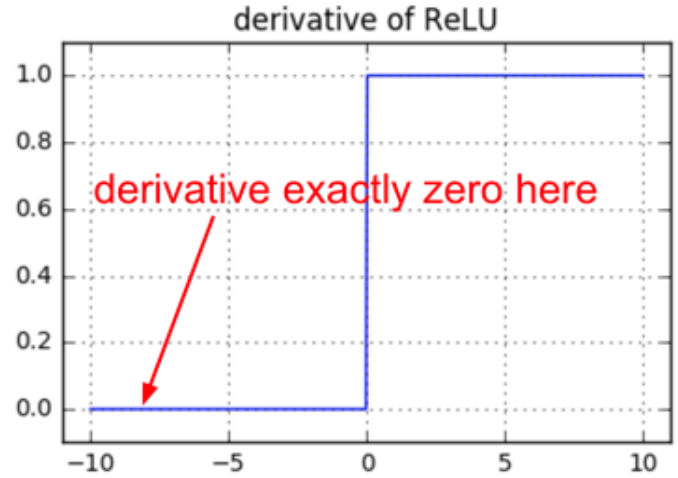


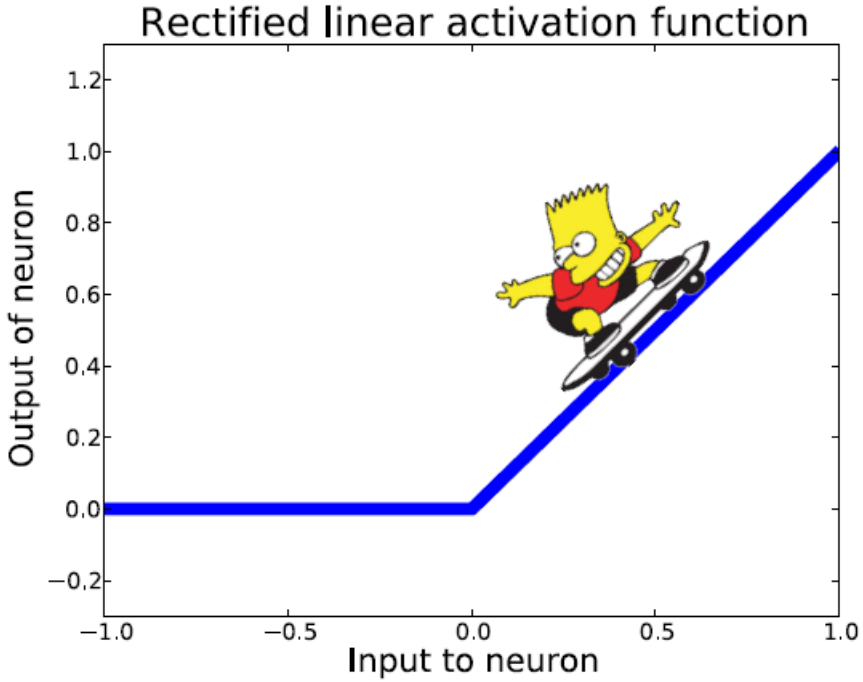
From Sigmoid to ReLU





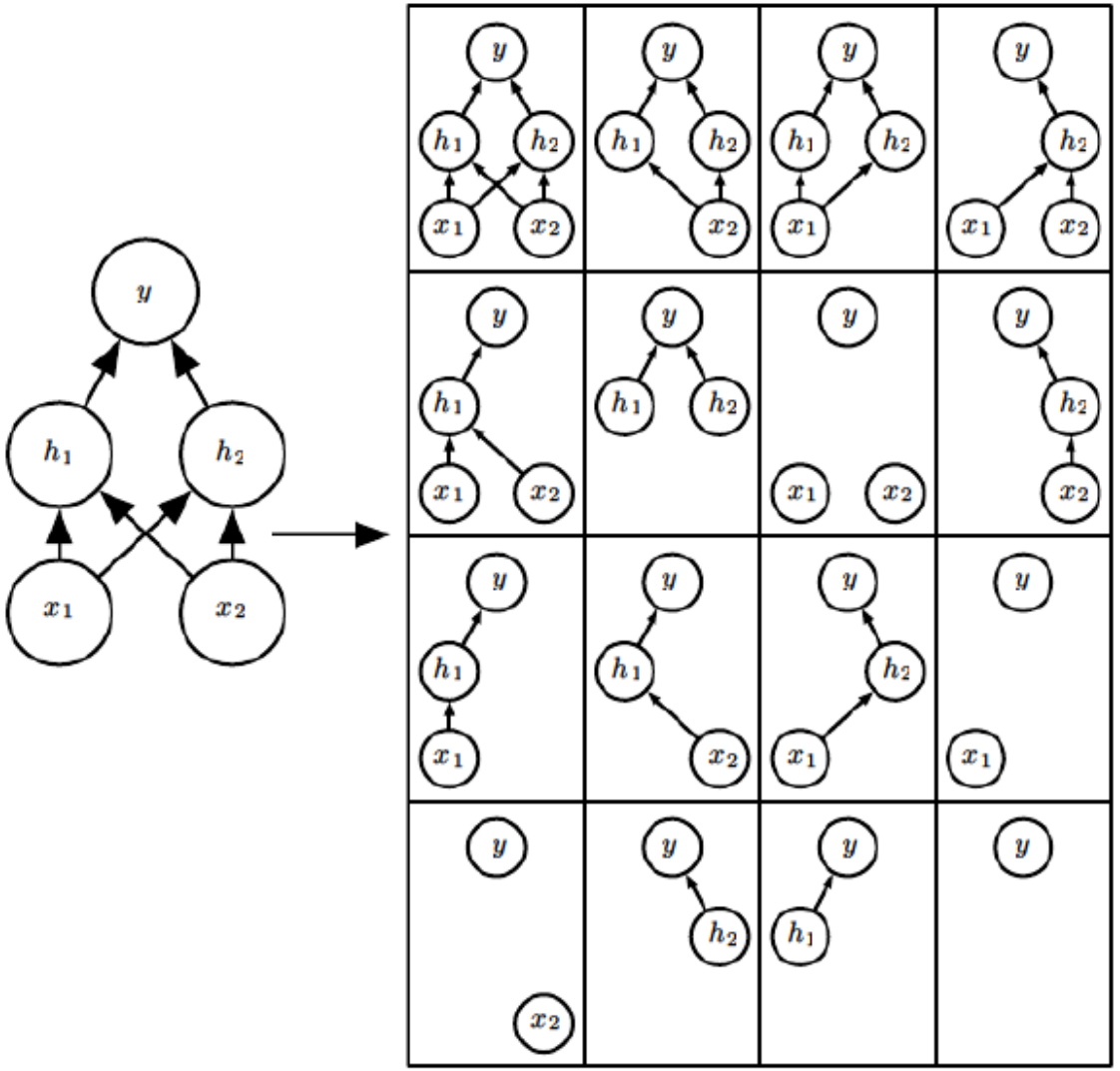




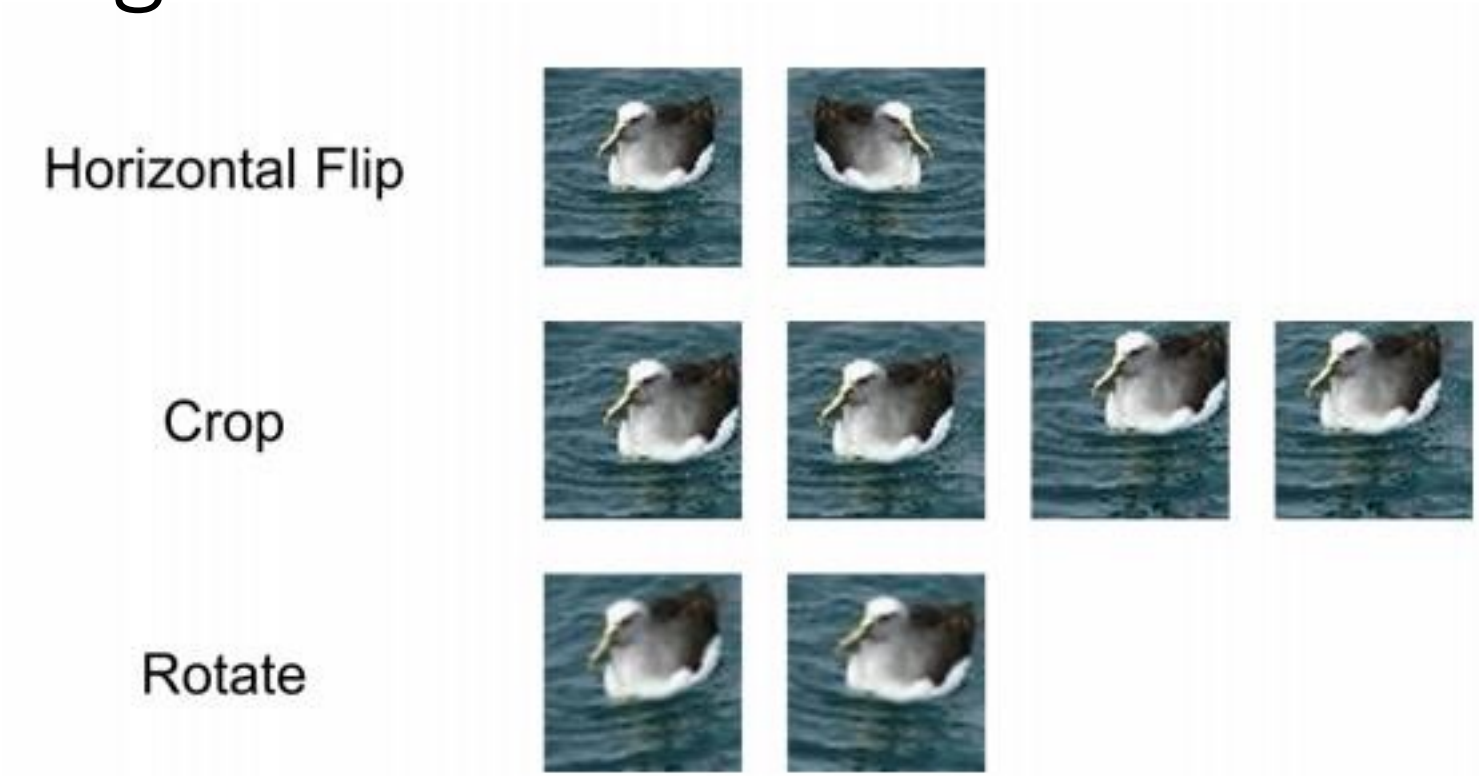


Dropout

- Randomly select weights to update
  - In each update step, randomly sample a different binary mask to all the input and hidden units
  - Multiple the mask bits with the units and do the update as usual
  - Typical dropout probability: 0.2 for input and 0.5 for hidden units
  - Very useful for FC layers, less for conv layers, not useful in RNNs



Data Augmentation



- Adding noise to the input: a special kind of augmentation
- Be careful about the transformation applied -> label preserving
  - **Example:** classifying ‘b’ and ‘d’; ‘6’ and ‘9’

VGG-Net, 2014

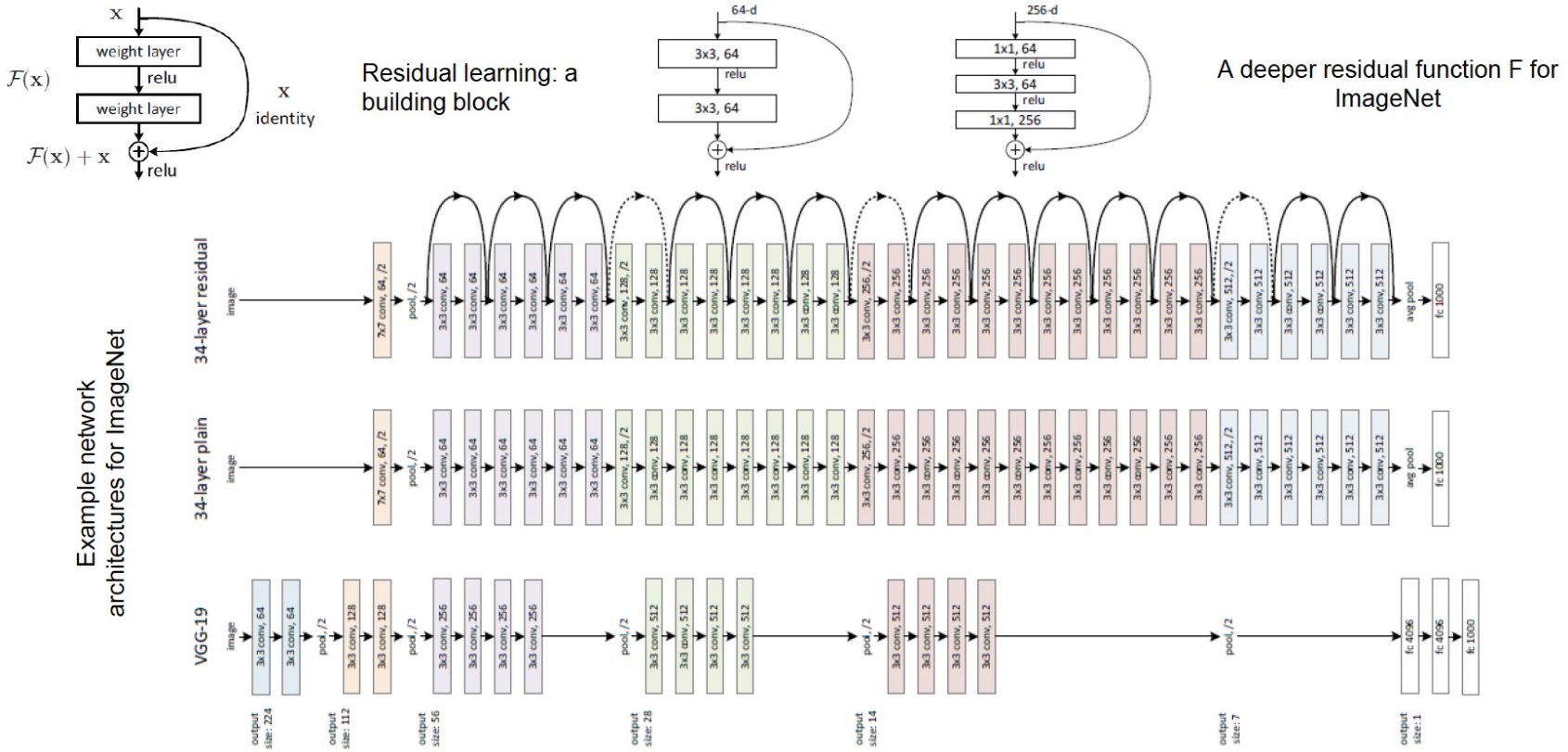
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Key Technical Features:

- Increase depth (up to 19)
- Smaller filter size (3)

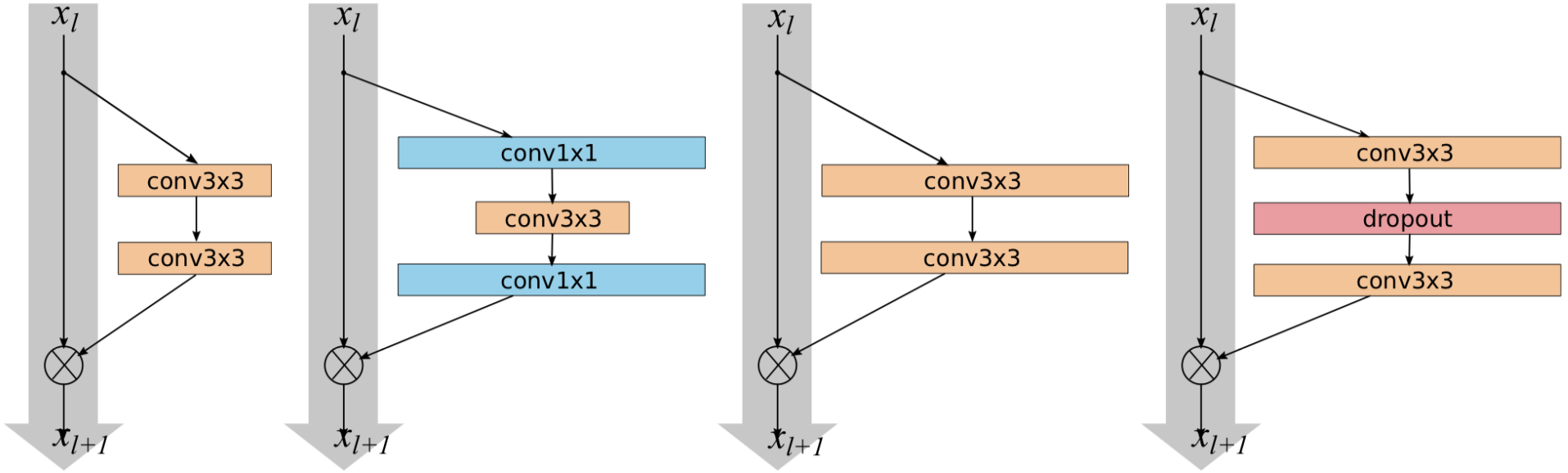
Configurations D and E are widely used for various tasks, called *VGG-16* and *VGG-19*

Deep Residual Network (ResNet), 2015



Key Technical Features: skip connections for residual mapping, up to > 1000 layers

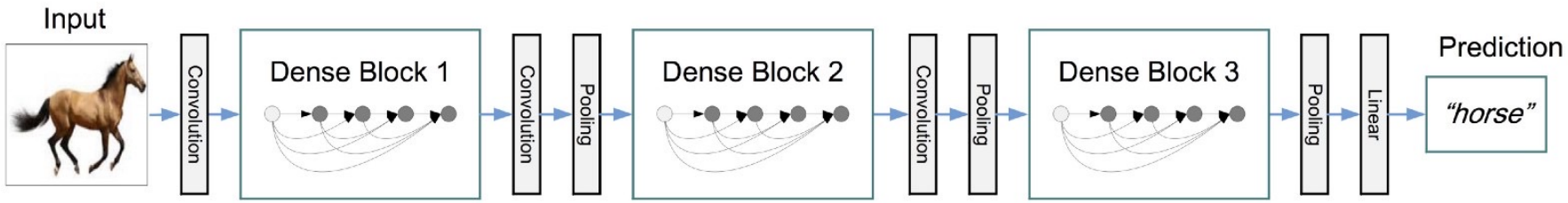
Wide ResNet, 2016

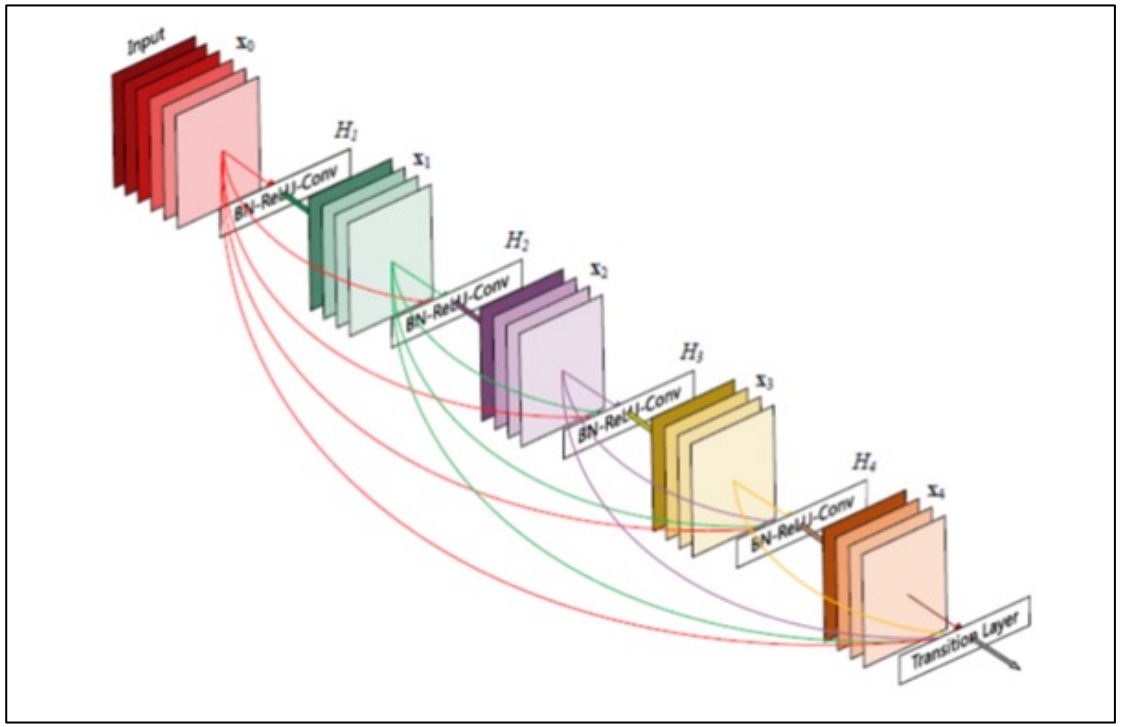


(a) basic (b) bottleneck (c) basic-wide (d) wide-dropout

- Widening of ResNet blocks (if done properly) provides a more effective way of improving performance of residual networks compared to increasing their depth.
- A wide 16-layer deep network has the same accuracy as a 1000-layer thin deep network and a comparable number of parameters, although being several times faster to train.

Densely Connected Convolutional Networks (DenseNet), 2017





Key Technical Features:

- Finer combination of multi-scale features (or whatever…)

ResNext, 2017

- Core idea: **multi-path** structure
  - **“Split-transform-merge”** strategy
  - New notion of “cardinality”
  - ResNet could be viewed as cardinality = 1

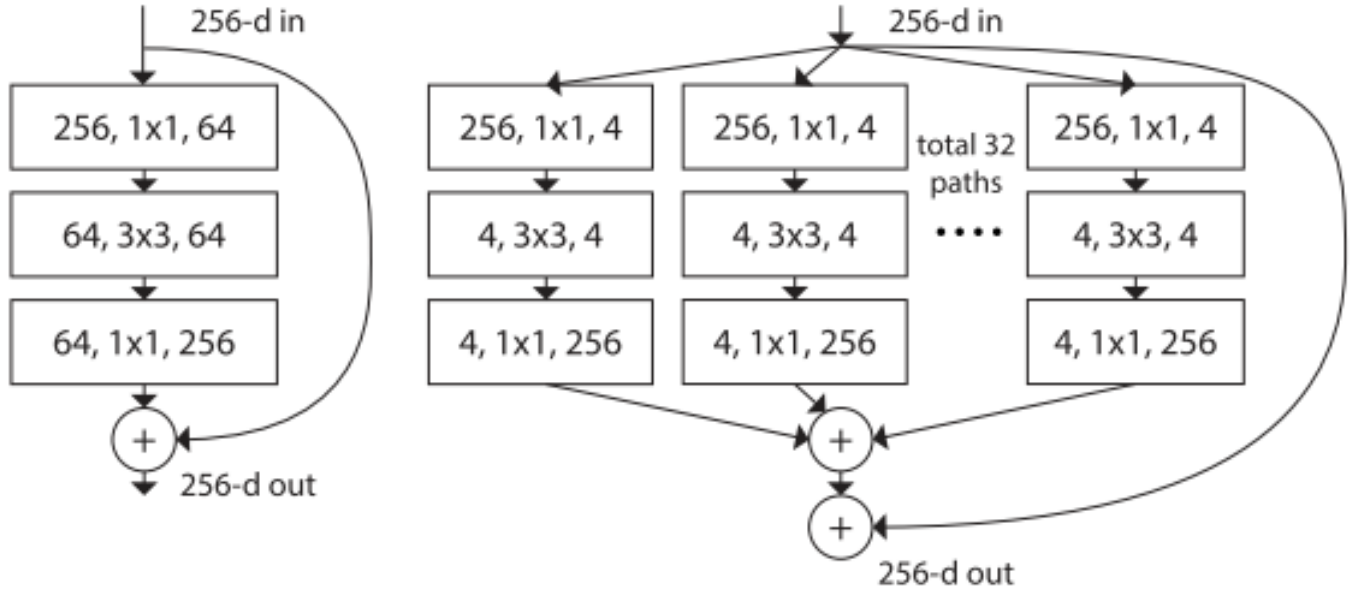


Figure 1. **Left**: A block of ResNet [14]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

Gradient Descent (GD)

Algorithm 1 Batch Gradient Descent at Iteration k

Require: Learning rate ϵ_k

Require: Initial Parameter θ

1: **while** stopping criteria not met **do**

2: Compute gradient estimate over N examples:

3: $\hat{\mathbf{g}} \leftarrow +\frac{1}{N}\nabla_\theta \sum_i L(f(\mathbf{x}^{(i)};\theta), \mathbf{y}^{(i)})$

4: Apply Update: $\theta \leftarrow \theta - \epsilon\hat{\mathbf{g}}$

5: **end while**

- Positive: Gradient estimates are stable
- Negative: Need to compute gradients over the entire training for one update

Stochastic Gradient Descent (SGD)

Algorithm 2 Stochastic Gradient Descent at Iteration k

Require: Learning rate ϵ_k

Require: Initial Parameter θ

1: **while** stopping criteria not met **do**
2: Sample example $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$ from training set
3: Compute gradient estimate:
4: $\hat{\mathbf{g}} \leftarrow +\nabla_\theta L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$
5: Apply Update: $\theta \leftarrow \theta - \epsilon\hat{\mathbf{g}}$
6: **end while**

- ϵ_k is learning rate at step k
- Sufficient condition to guarantee convergence:

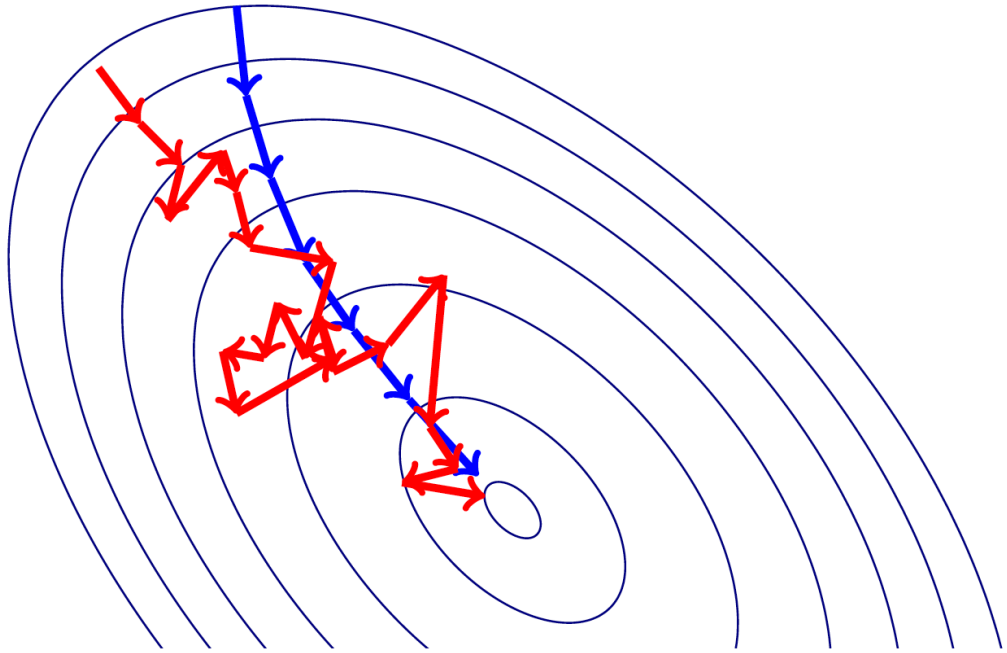
$$\sum_{k=1}^{\infty} \epsilon_k = \infty \text{ and } \sum_{k=1}^{\infty} \epsilon_k^2 < \infty$$

GD versus SGD

- Batch Gradient Descent:

$$\hat{\mathbf{g}} \leftarrow +\frac{1}{N}\nabla_\theta \sum_i L(f(\mathbf{x}^{(i)};\theta),\mathbf{y}^{(i)})$$

$$\theta \leftarrow \theta - \epsilon\hat{\mathbf{g}}$$



- SGD:

$$\hat{\mathbf{g}} \leftarrow +\nabla_\theta L(f(\mathbf{x}^{(i)};\theta),\mathbf{y}^{(i)})$$

$$\theta \leftarrow \theta - \epsilon\hat{\mathbf{g}}$$

Minibatch

- Potential Problem: Gradient estimates can be very noisy
- Obvious Solution: Use larger mini-batches (In theory, growingly larger)

- Advantage: Computation time per update does not depend on number of training examples.
- This allows convergence on extremely large datasets

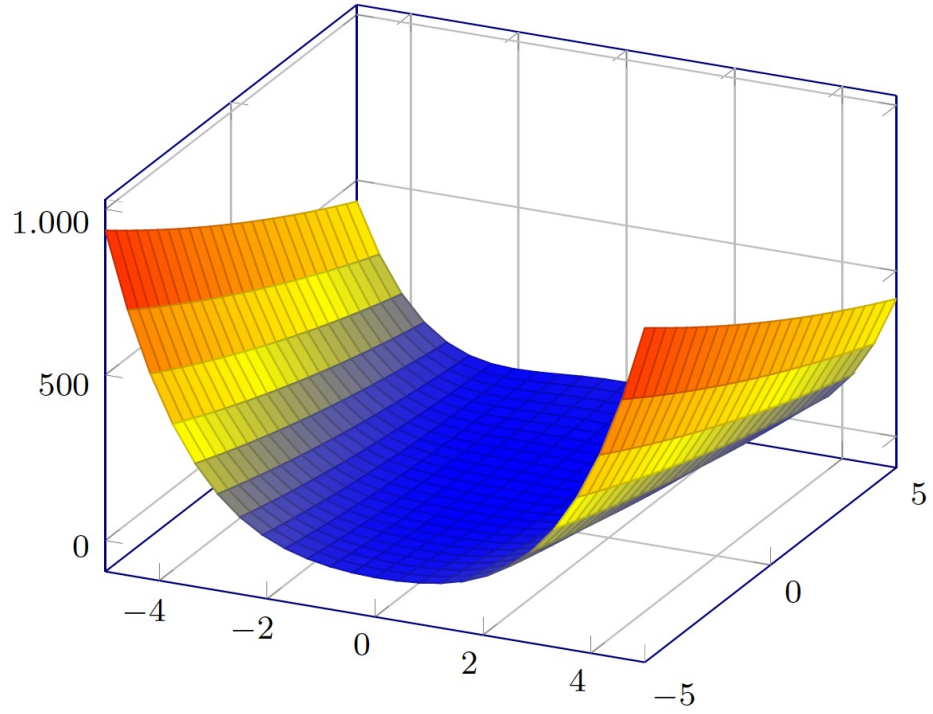
- **The larger MB size the better (only if you can)!!**

"Large Scale Learning with Stochastic Gradient Descent", Leon Bottou.

Momentum

- The Momentum method is a method to accelerate learning using SGD

- In particular SGD suffers in the following scenarios:
  - Error surface has high curvature
  - Small but consistent gradients
  - Noisy gradients



- Gradient Descent would move quickly down the walls, but very slowly through the valley floor

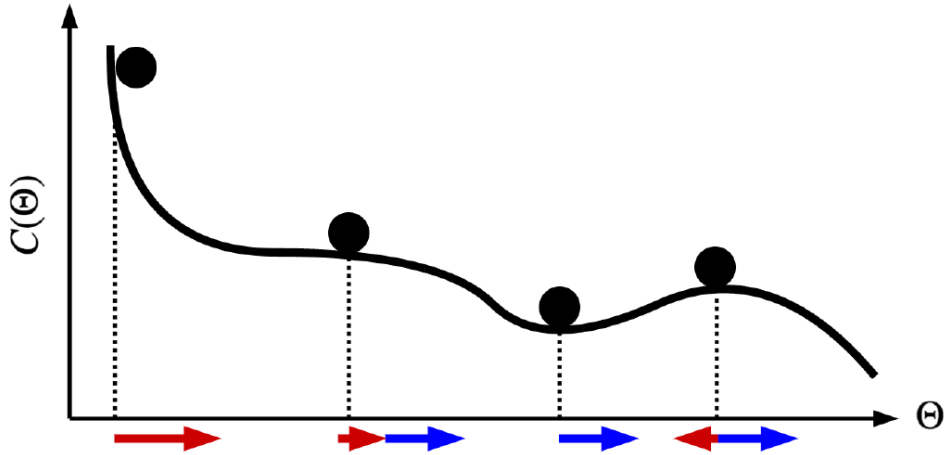
Momentum

- Update rule in SGD:

$$\Theta^{(t+1)} \leftarrow \Theta^{(t)} - \eta \boldsymbol{g}^{(t)}$$

where $\boldsymbol{g}^{(t)} = \nabla_{\Theta} C(\Theta^{(t)})$

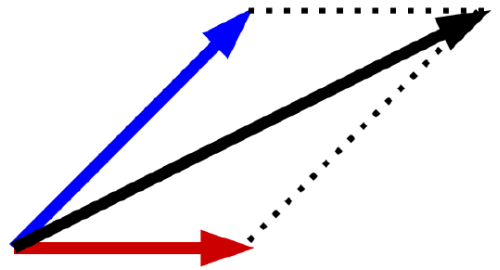
  - Gets stuck in local minima or saddle points



- Momentum: make the same movement $\boldsymbol{v}^{(t)}$ in the last iteration, corrected by negative gradient:

$$\boldsymbol{v}^{(t+1)} \leftarrow \lambda \boldsymbol{v}^{(t)} - (1-\lambda)\boldsymbol{g}^{(t)}$$

$$\Theta^{(t+1)} \leftarrow \Theta^{(t)} + \eta \boldsymbol{v}^{(t+1)}$$



- $\boldsymbol{v}^{(t)}$ is a moving average of $-\boldsymbol{g}^{(t)}$

Adaptive Learning Rate Optimization

- Popular Solver Examples: AdGrad, RMSProp, Adam

SGD: $\theta \leftarrow \theta - \epsilon \hat{\mathbf{g}}$

Momentum: $\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \hat{\mathbf{g}}$ then $\theta \leftarrow \theta + \mathbf{v}$

Nesterov: $\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \nabla_\theta \left(L(f(\mathbf{x}^{(i)}; \theta + \alpha \mathbf{v}), \mathbf{y}^{(i)}) \right)$ then $\theta \leftarrow \theta + \mathbf{v}$

AdaGrad: $\mathbf{r} \leftarrow \mathbf{r} + \mathbf{g} \odot \mathbf{g}$ then $\Delta\theta - \leftarrow \frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \mathbf{g}$ then $\theta \leftarrow \theta + \Delta\theta$

RMSProp: $\mathbf{r} \leftarrow \rho \mathbf{r} + (1 - \rho) \hat{\mathbf{g}} \odot \hat{\mathbf{g}}$ then $\Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \hat{\mathbf{g}}$ then $\theta \leftarrow \theta + \Delta\theta$

Adam: $\hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1 - \rho_1^t}, \hat{\mathbf{r}} \leftarrow \frac{\mathbf{r}}{1 - \rho_2^t}$ then $\Delta\theta = -\epsilon \frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}}} + \delta}$ then $\theta \leftarrow \theta + \Delta\theta$

Batch Normalization

- In ML, we assume future data will be drawn from same probability distribution as training data
- For a hidden layer, after training, the earlier layers have new weights and hence may generate a new distribution for the next hidden layer
- We want to reduce this internal covariate shift for the benefit of later layers

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Monitor Your Training Curve

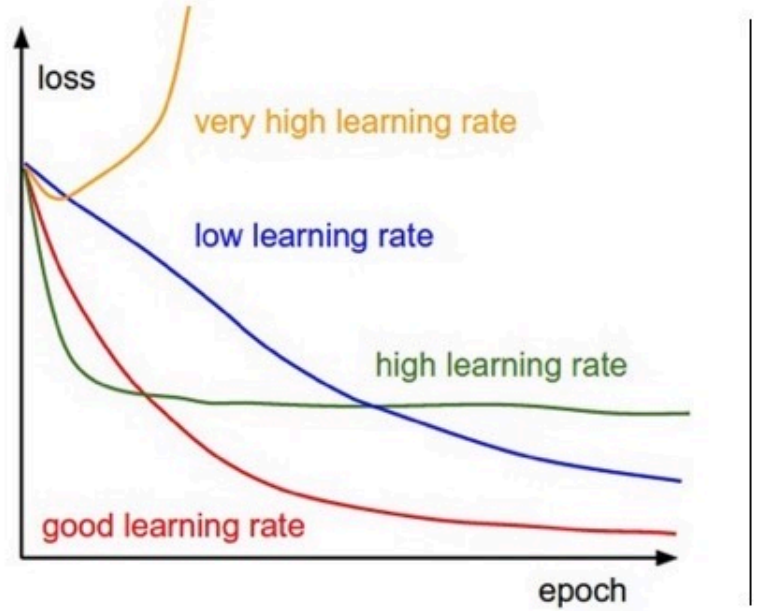


Figure 1

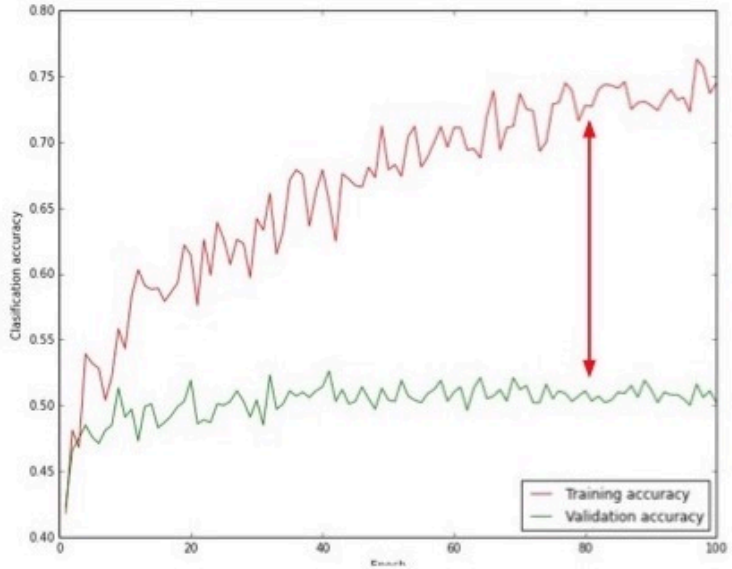


Figure 3

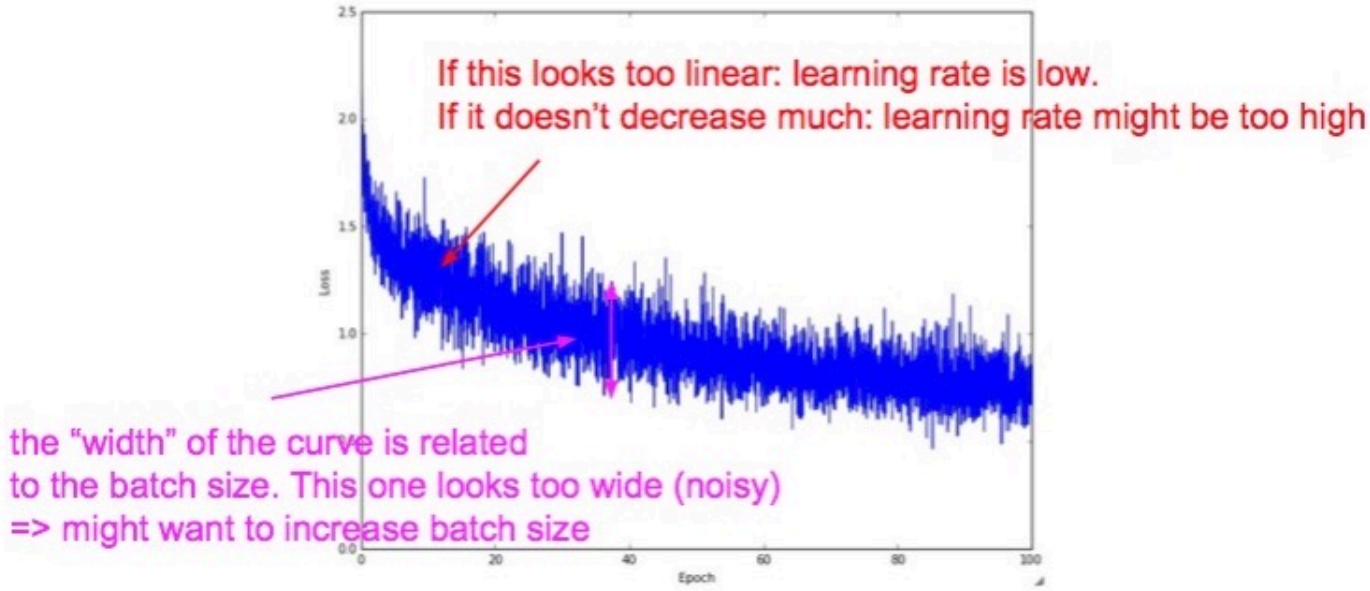
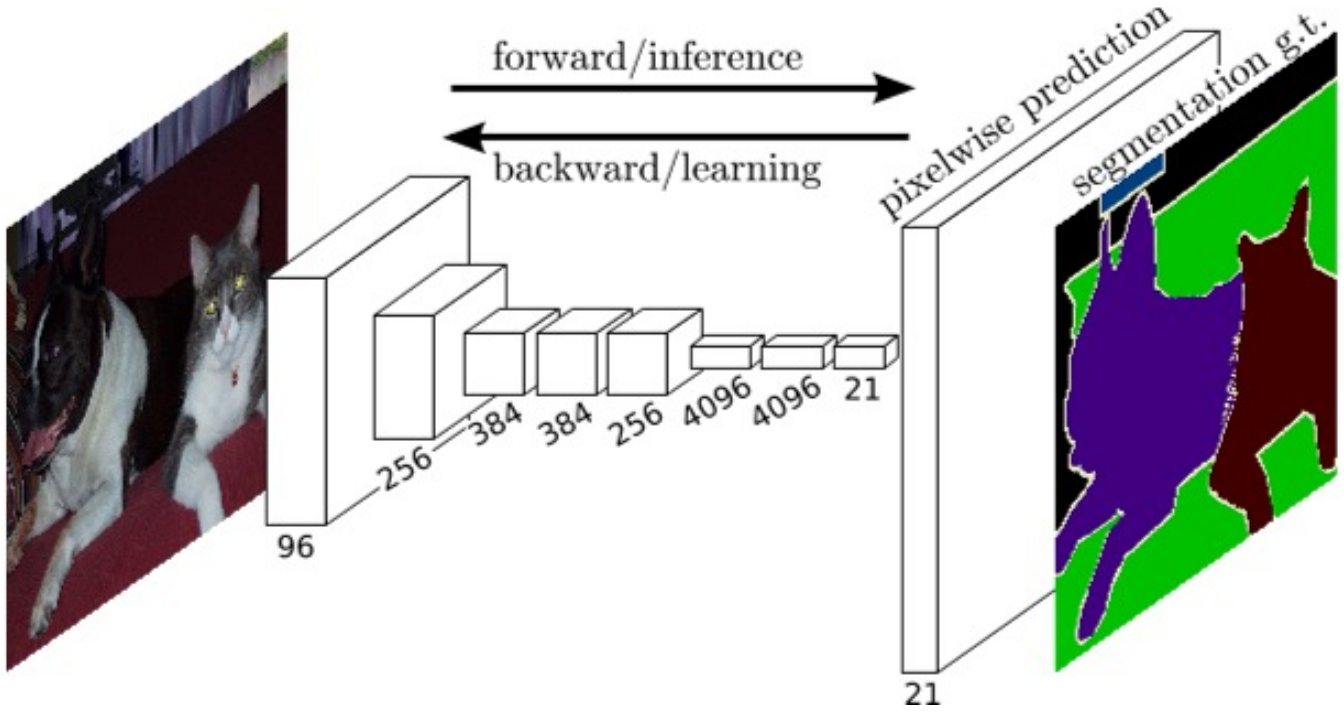


Figure 2

(More) Art of Convolutions

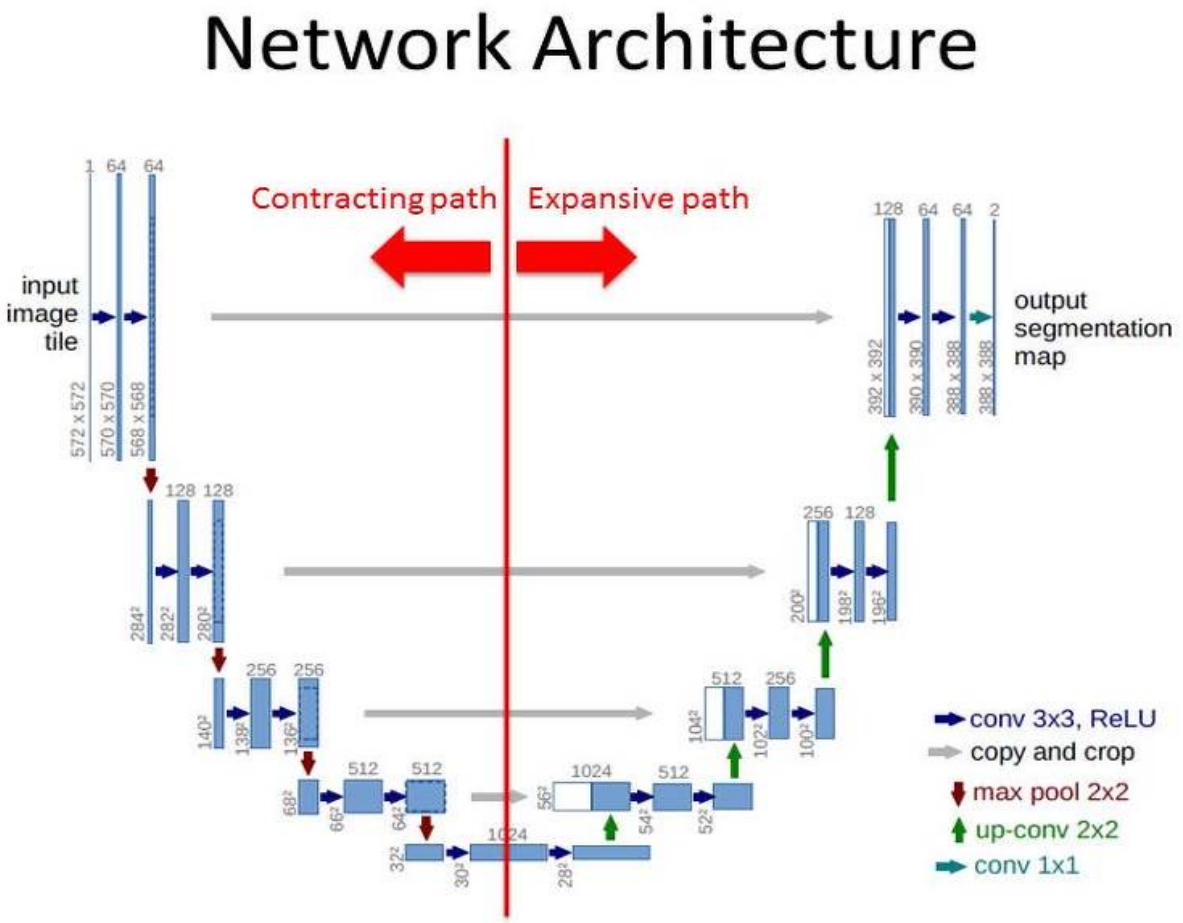
Fully Convolutional Network (FCN), 2014



Key Technical Features:

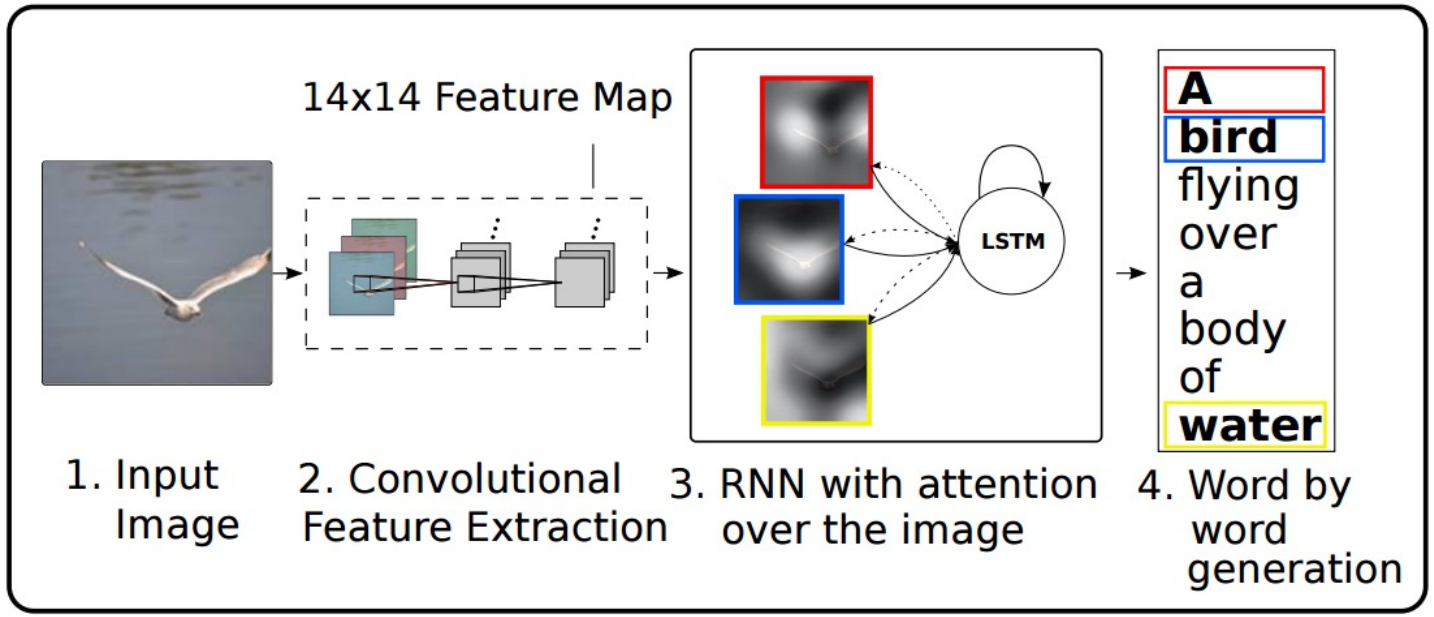
- No fully-connected layer -> No fixed requirement on input size
- Widely adopted in pixel-to-pixel prediction tasks, e.g., image segmentation

U-Net, 2015



- The architecture consists of a contracting path to capture context
- …and a symmetric expanding path to enable precise localization.
- Also fully convolutional
- Very popular backbone for dense prediction (image segmentation, restoration…)

Attention Mechanism



"Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", 2015

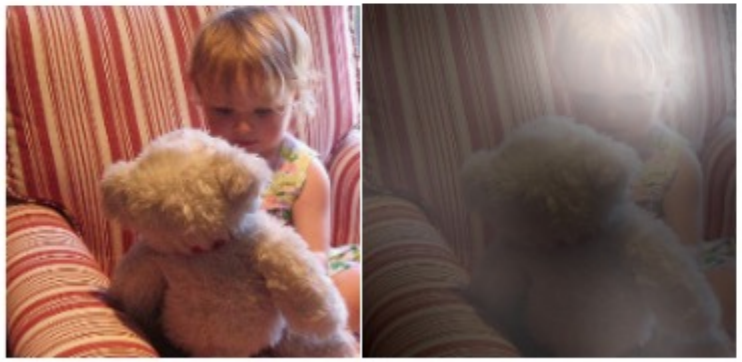
- **Idea is simple:** add a (learned) weighted mask to feature (feature selection)
- Use a feed-forward deep network to extract L feature vectors
- Can use a recurrent network to iteratively update the attention (shown as bright regions) for each output word
- Find meaningful correspondences between words and attentions

Examples of (Input) Visual Attention

A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

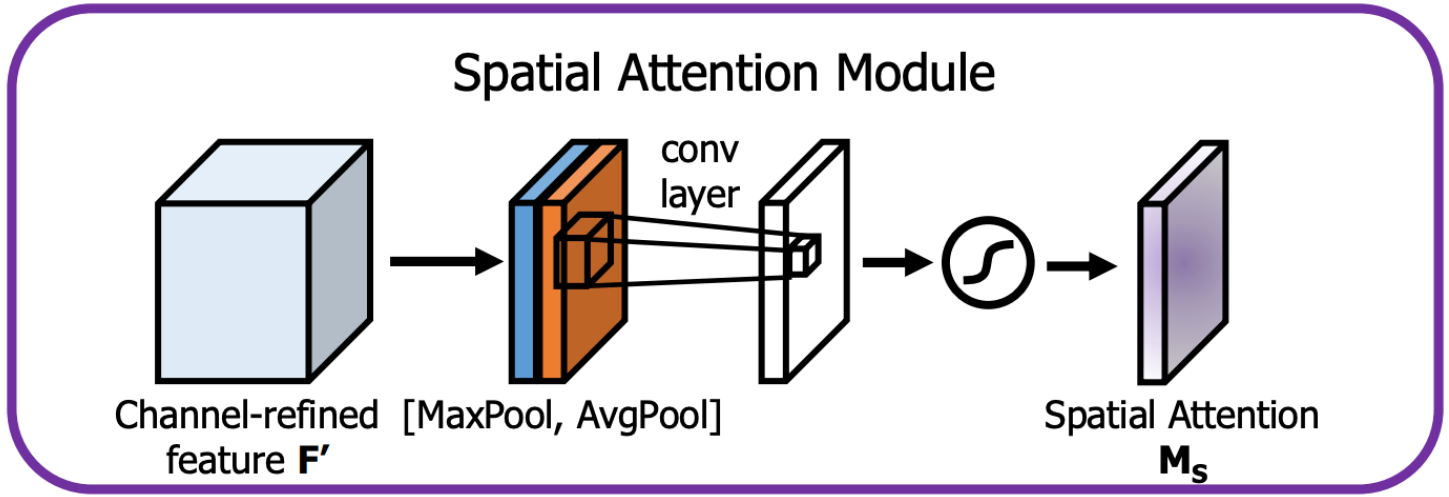


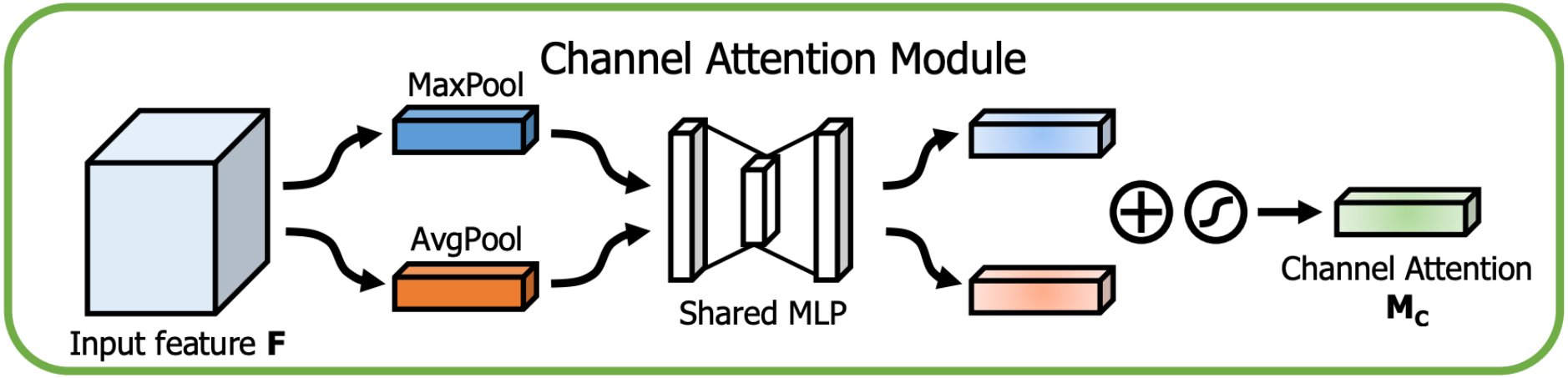
A little girl sitting on a bed with a teddy bear.

A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

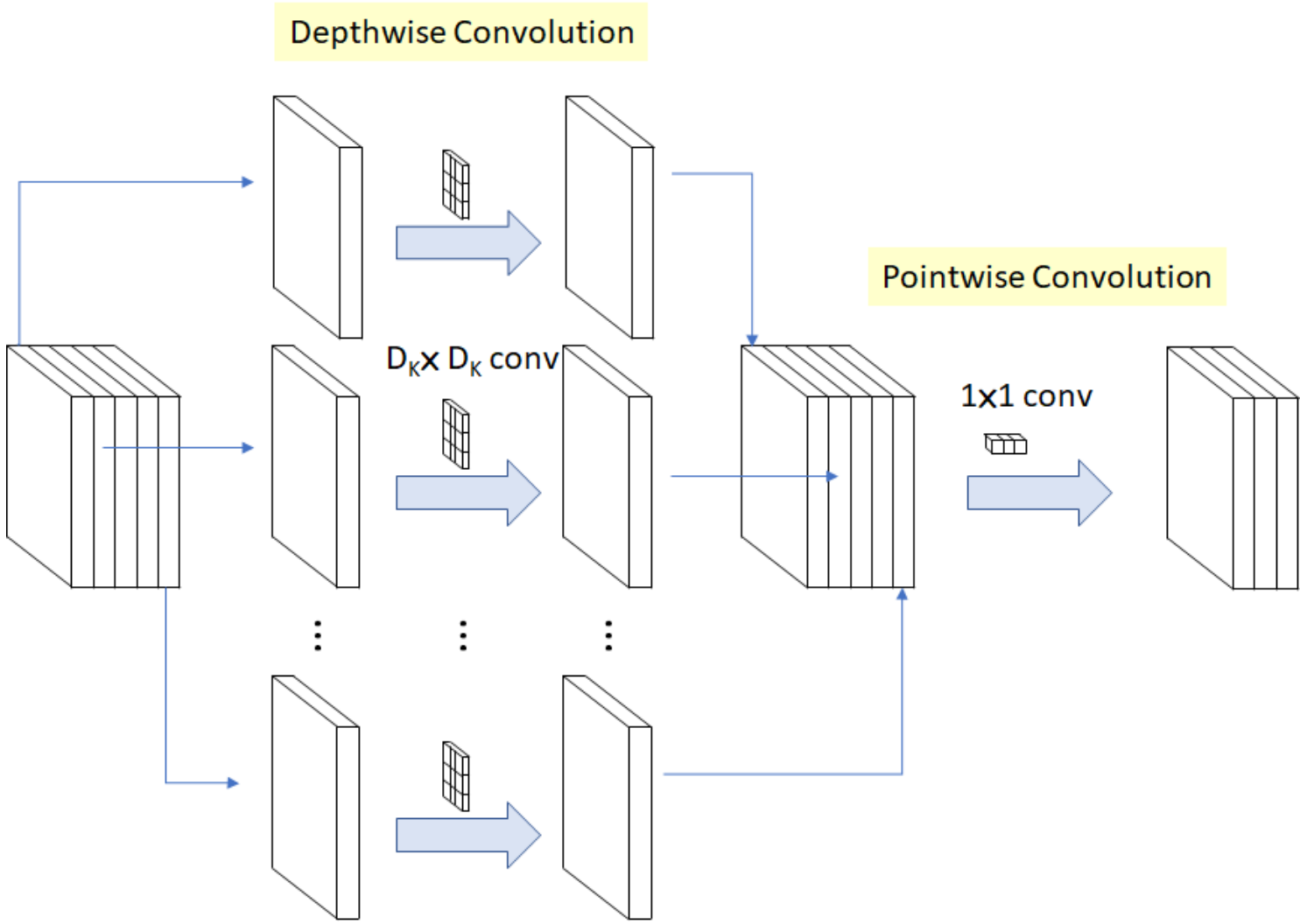
Spatial and Channel Attention





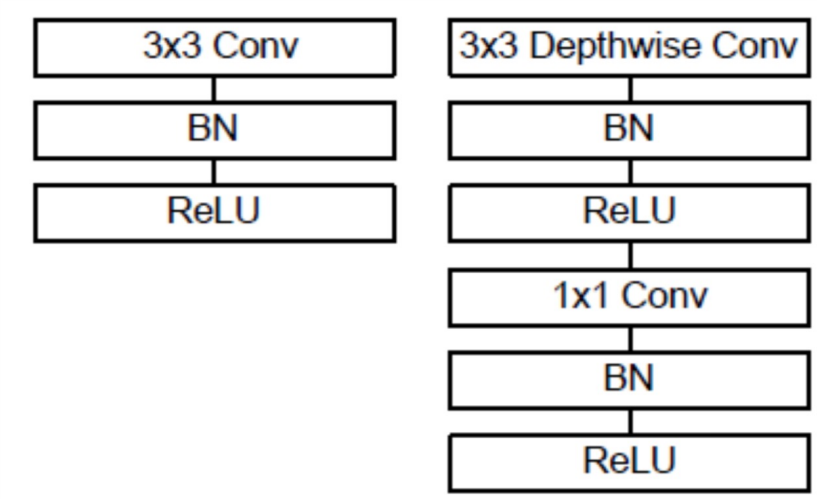
Depth-Wise Convolution

- **Depthwise convolution** is the channel-wise spatial convolution.
- It is often used together with **pointwise convolution,** i.e., 1×1 convolution to change the channel dimension (number of feature maps)



MobileNet (v1)

- Single streamlined, very light-weight architecture
- **Main idea:** Depthwise Separable Convolutions
- **Other ideas:** Width Multiplier α for Thinner Models + Resolution Multiplier ρ for Reduced Representation



Standard Convolution (Left), Depthwise separable convolution (Right) With BN and ReLU

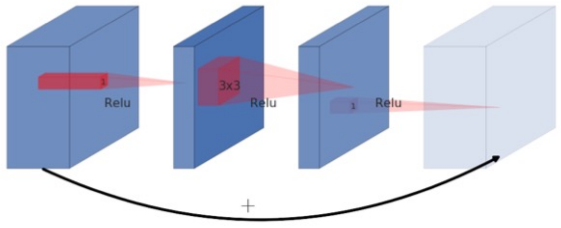
Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

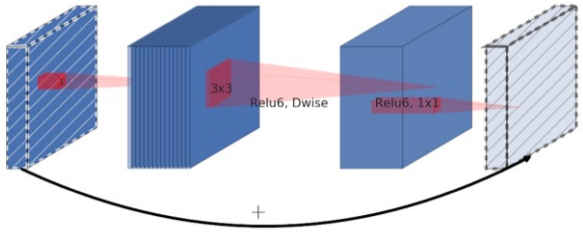
MobileNet (v2)

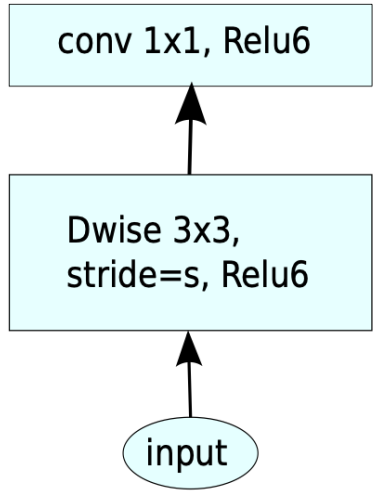
- **Main idea:** inverted residual structure
  - Adding residual connections between the narrow bottleneck layers (considerably more memory efficient - **Why?**)
  - Non-linearities are removed in narrow layers to maintain representational power
  - The intermediate expansion layer uses lightweight depthwise convolutions to filter features as a source of non-linearity

(a) Residual block

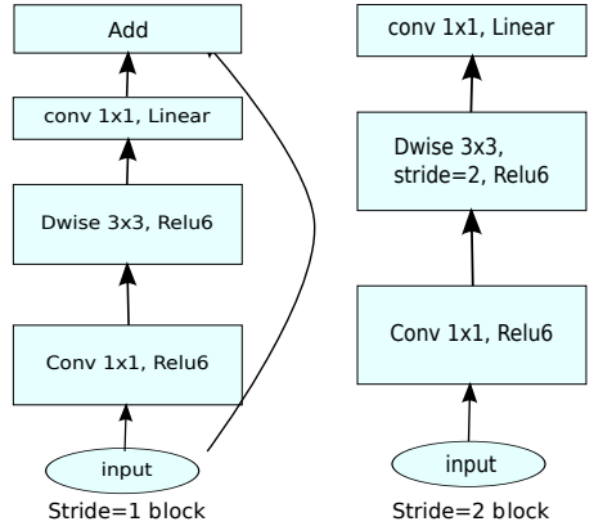


(b) Inverted residual block



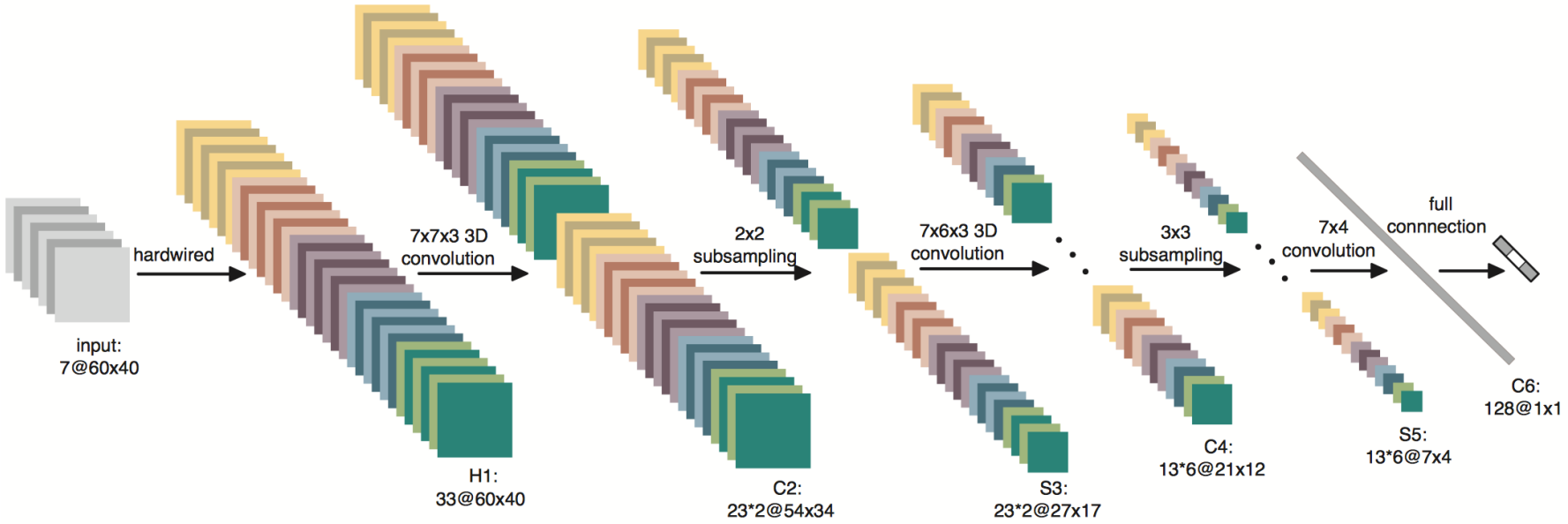


(b) MobileNet[27]



(d) Mobilenet V2

3D Convolutional Network (3D CNN), 2011

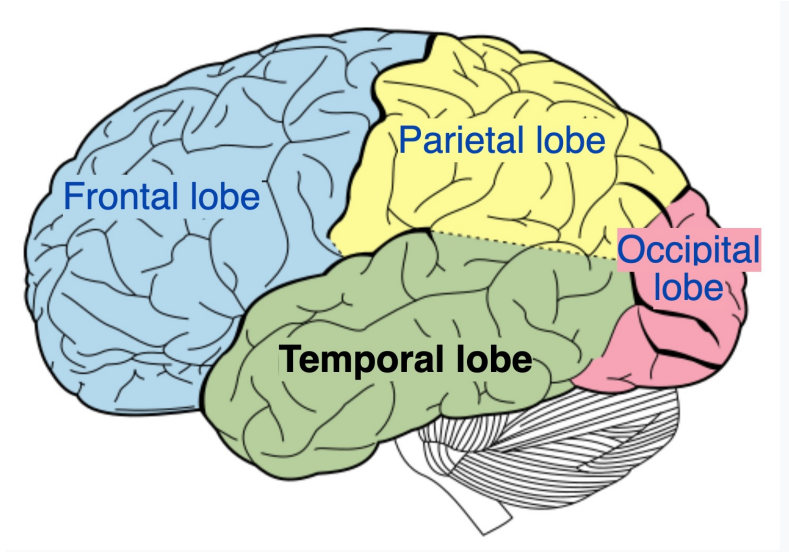


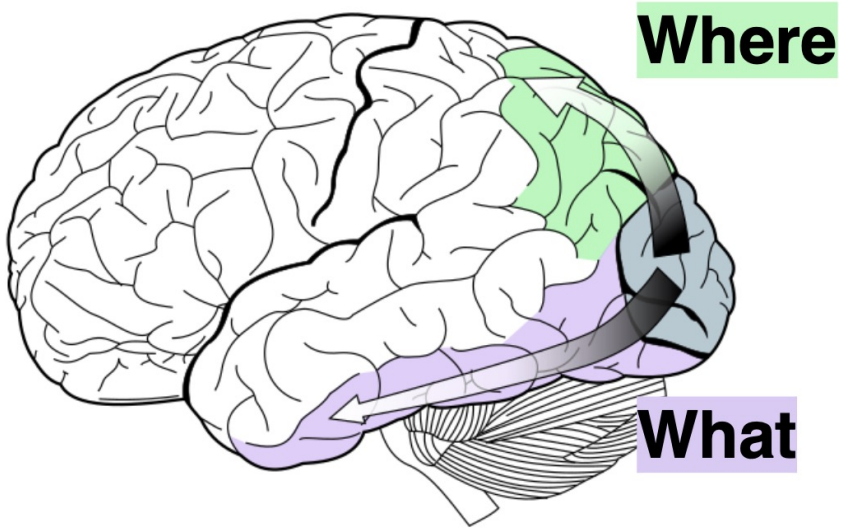
Key Technical Features:

- Going from 2D convolutional filters to 3D filters, to take temporal coherence into consideration

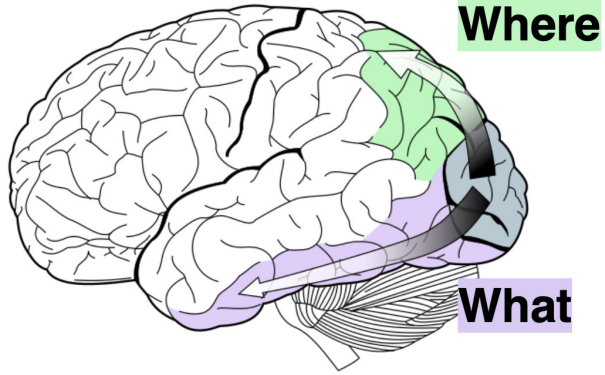
More Efficient Design?

- “**Two-streams hypothesis**” for human vision
  - The **dorsal stream** involves in the guidance of actions and recognizing where objects are in space. It contains a detailed map of the visual field. and detects & analyzes location movements
  - The **ventral stream** is associated with object recognition and form representation. Also described as the “what” stream, it has strong connections to the dorsal stream and other brain regions controlling memory or emotion
- **Long story short:** human brains use two relatively independent systems to recognize objects and to record temporal movements.





Two Stream Network, 2014



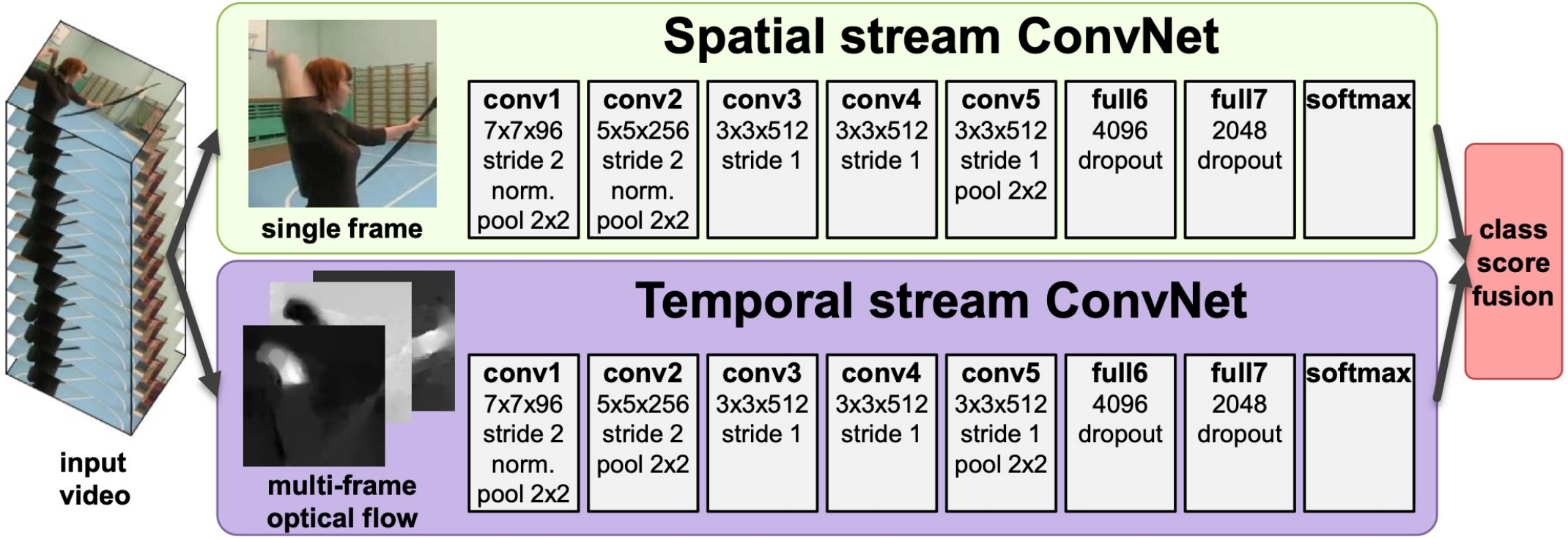
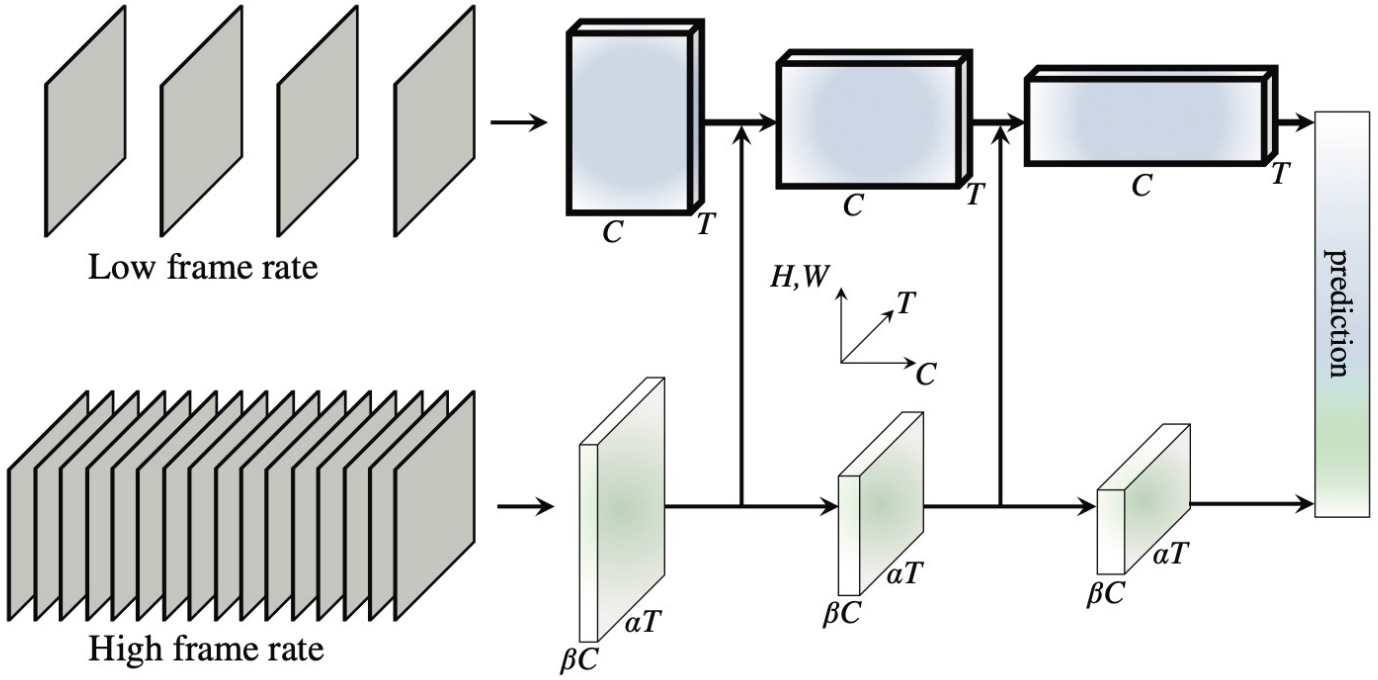


Figure 1: **Two-stream architecture for video classification.**

Slow-Fast Network, 2019

A state-of-the-art two-stream model with

- *(i) a Slow pathway, operating at low frame rate, to capture spatial semantics*
- *(ii) a Fast pathway, operating at high frame rate, to capture motion at fine temporal resolution.*



Why Transformer for Vision?

- Towards a **general, conceptual simple**, and **sufficiently versatile** architecture yet still achieving competitive performance for vision?
- The inductive bias of CNNs, e.g., spatially invariant and locality-based, also may not be sufficient …

Basics: Transformer in NLP

- Standard model in NLP tasks
- Only consists of self-attention modules, instead of RNN
- Encoder-decoder
- Requires large dataset and high computational cost
- Pre-training and fine-tuning approaches : BERT & GPT

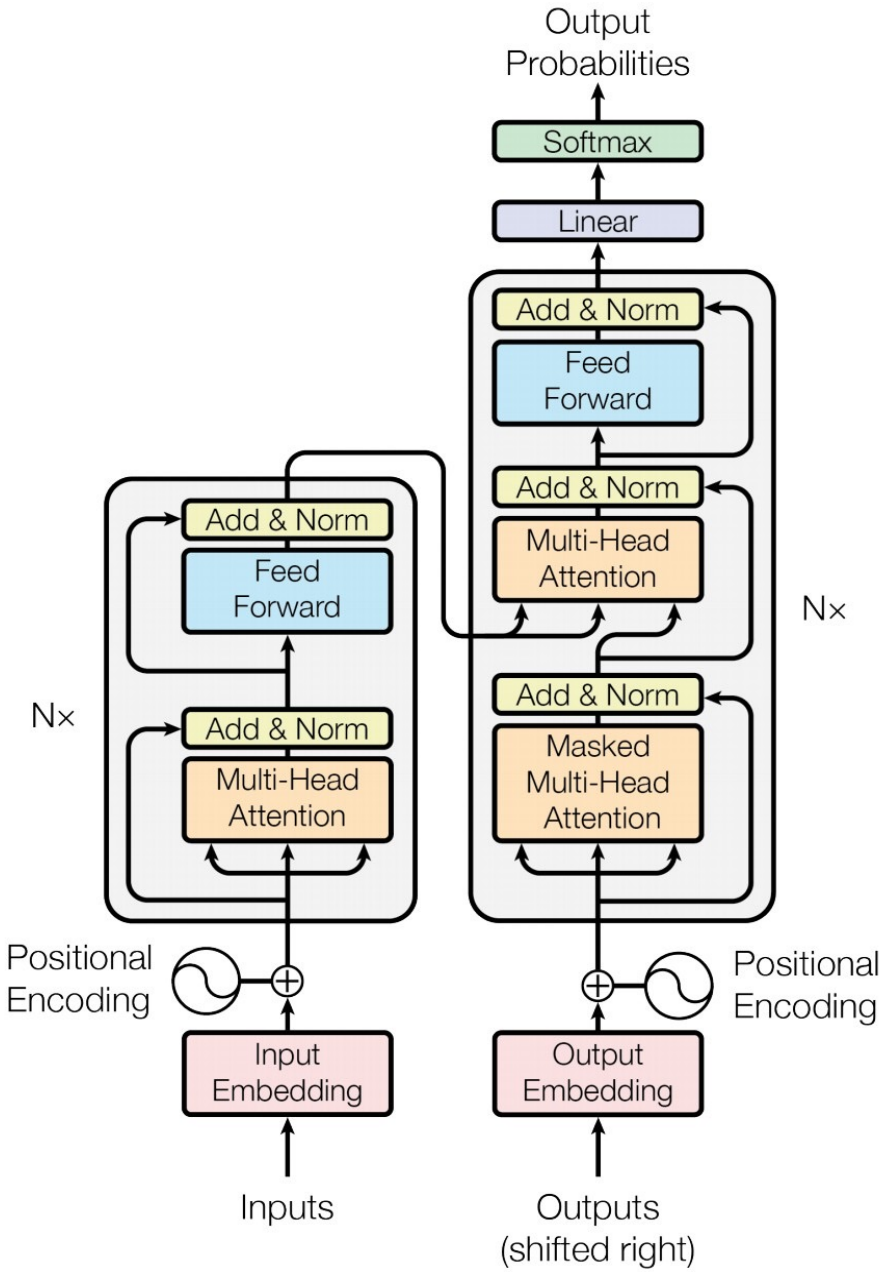
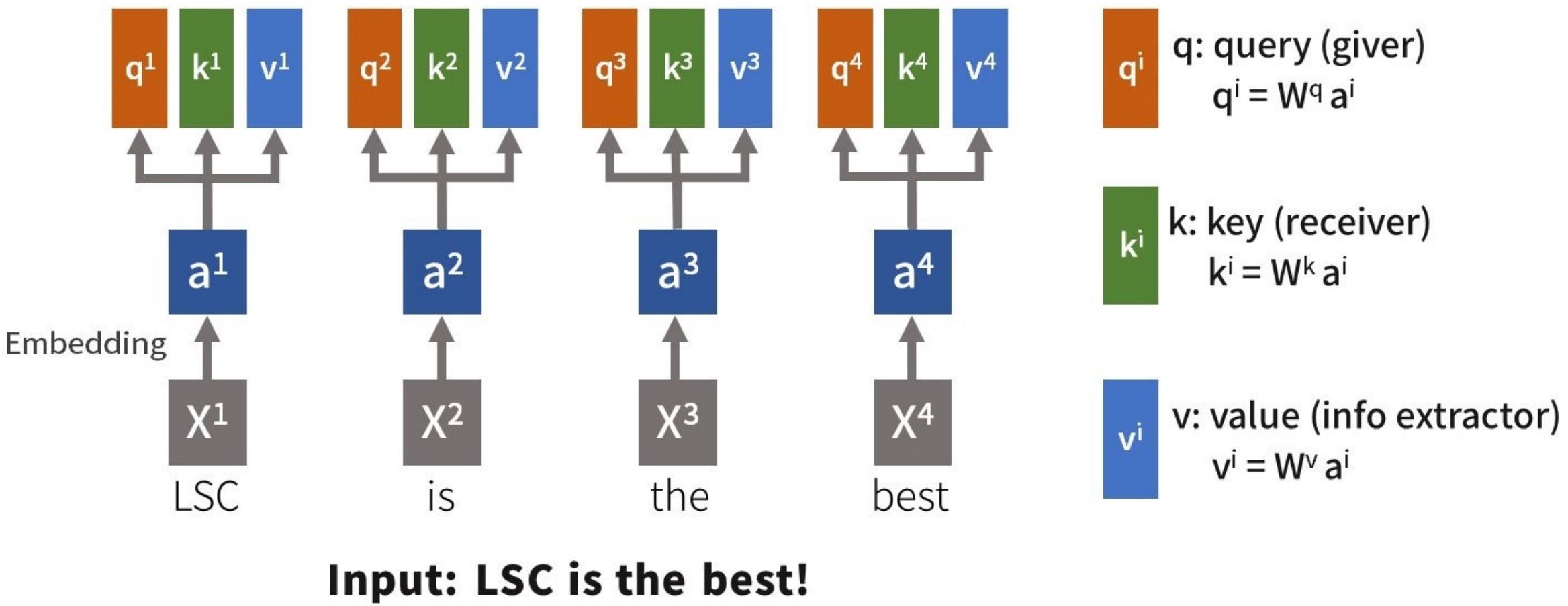
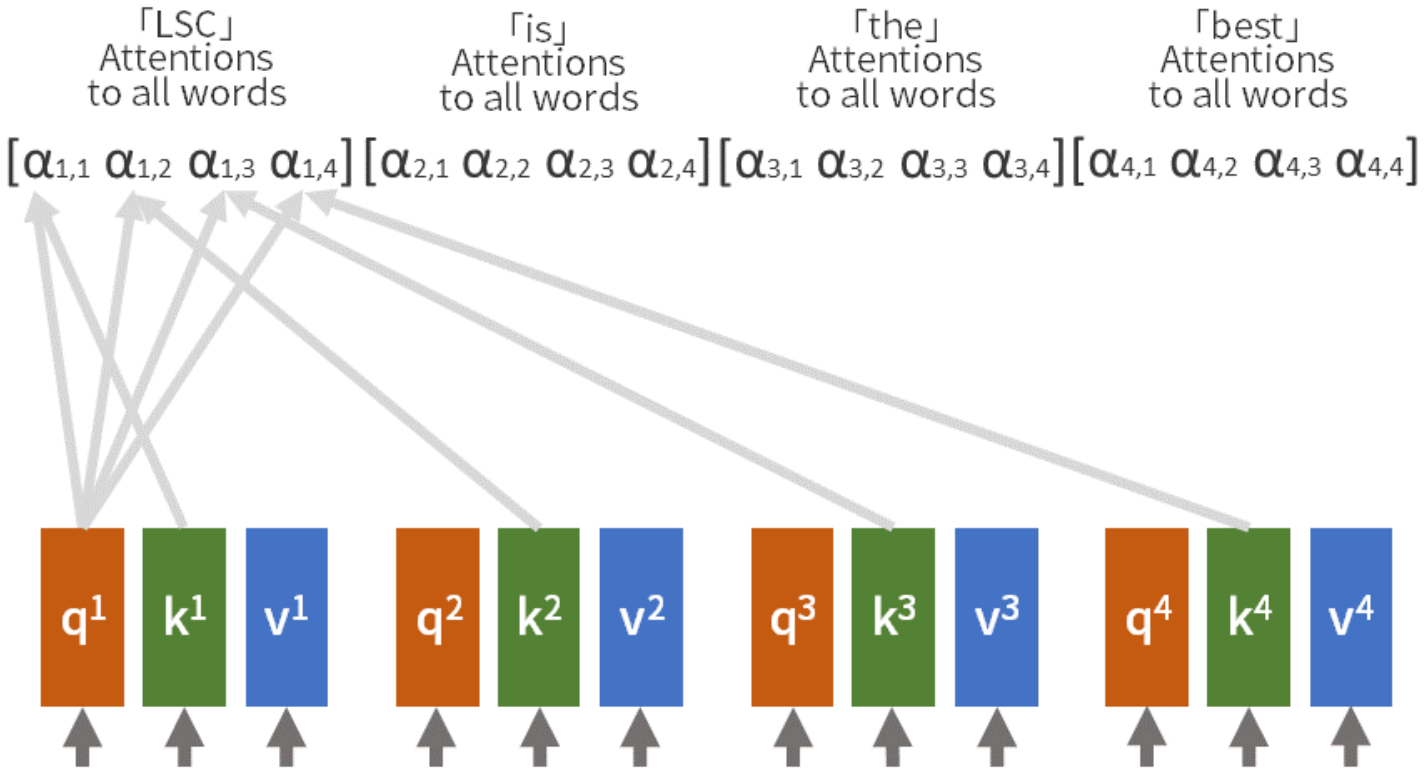


Figure 1: The Transformer - model architecture.

Basics: Self-Attention



Basics: Self-Attention



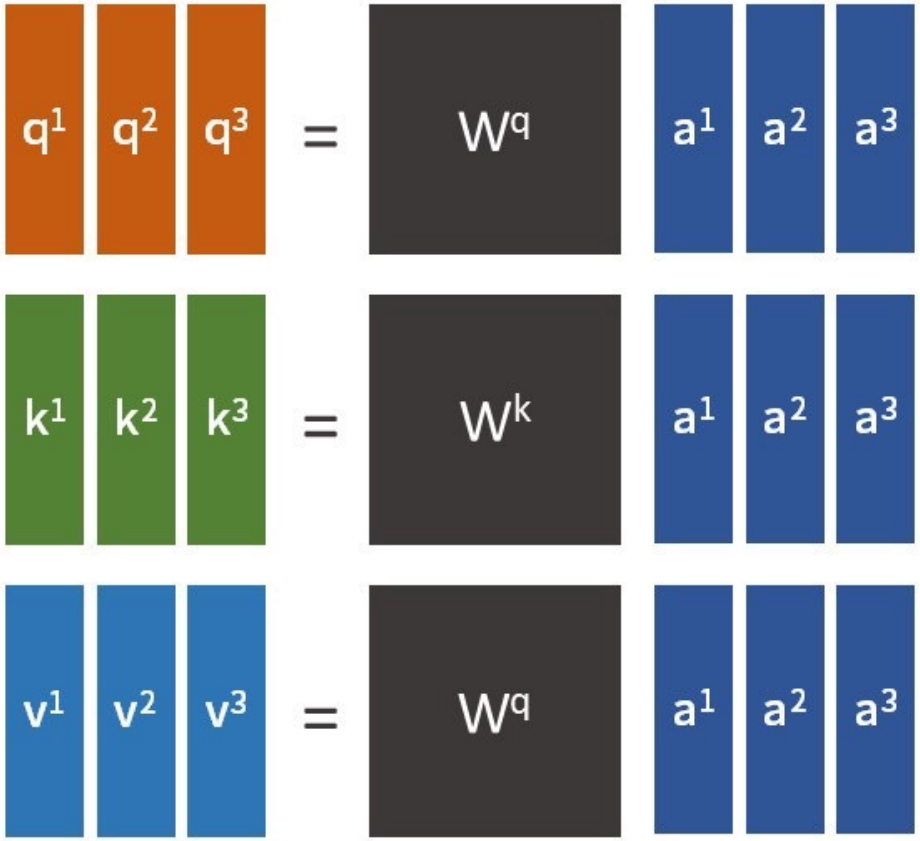
$$\alpha_{i,j} = \frac{q^i \cdot k^j}{\sqrt{d}}$$

d: dimension of q, k

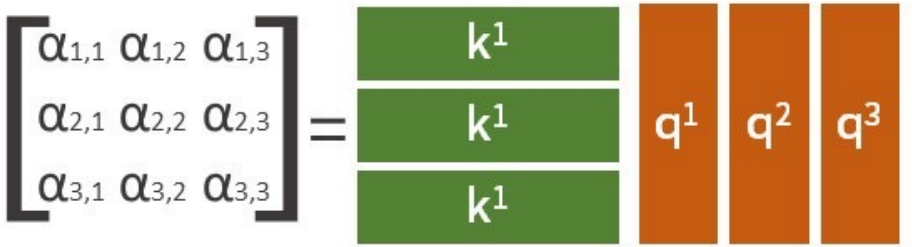
$$A = \begin{bmatrix} \alpha_{1,1} & \alpha_{1,2} & \alpha_{1,3} & \alpha_{1,4} \\ \alpha_{2,1} & \alpha_{2,2} & \alpha_{2,3} & \alpha_{2,4} \\ \alpha_{3,1} & \alpha_{3,2} & \alpha_{3,3} & \alpha_{3,4} \\ \alpha_{4,1} & \alpha_{4,2} & \alpha_{4,3} & \alpha_{4,4} \end{bmatrix}$$

Attention Matrix

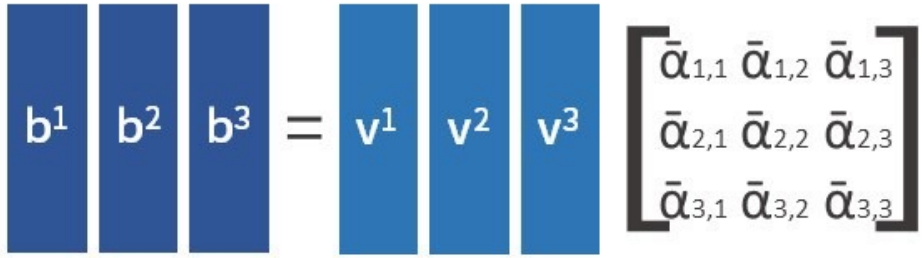
Basics: Self-Attention



Attention A:



Output:



Bringing Transformers into Computer Vision

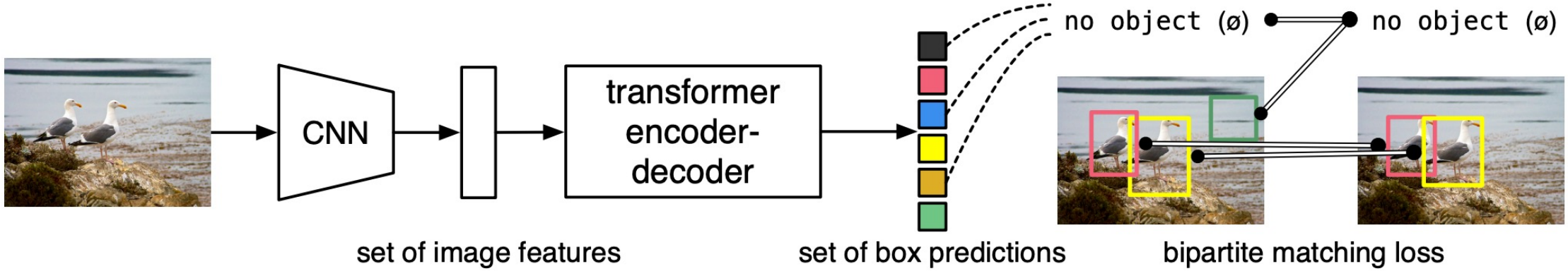
- Only in local neighborhoods (1)
  - Image Transformer, ICML 2018
  - Stand-alone self-attention in vision models, NeurIPS 2019
  - On the relationship between self-attention and convolutional layers, ICLR 2020
  - Exploring self-attention for image recognition, CVPR 2020
- Scalable approximations to global self-attention (2)
  - Generating long sequences with sparse transformers, arXiv 2019
- Blocks of varying sizes (3)
  - Scaling autoregressive video models, ICLR 2019
- Only along individual axes (4)
  - Axial attention in multidimensional transformers, arXiv 2019
  - Axial-deeplab: Stand-alone axial-attention for panoptic segmentation, ECCV 2020

Bringing Transformers into Computer Vision

- Combining CNN with self-attention (5)
  - Attention augmented convolutional networks, ICCV 2019, image classification
  - End-to-end object detection with transformers, ECCV 2020, object detection
  - Videobert: A joint model for video and language representation learning, ICCV 2019, video processing
  - Visual transformers, arxiv 2020, image classification
- Unified text-vision tasks (6)
  - VQA
  - Image Retrieval
  - OCR (Document Layout Analysis)
- Most Related Works (7)
  - Generative pretraining from pixels (iGPT), ICML 2020
  - Big Transfer (BiT): General Visual Representation Learning, ECCV 2020

DETR: End-to-End Object Detection with Transformers (ECCV'20)

- DETR directly predicts (in parallel) the final set of detections by combining a common CNN with a transformer architecture. It does **NOT** rely on the many hand-designed components like in FasterRCNN.



- **The takeaway from DETR is bi-folds:**
  - DETR achieved comparable performance to Faster R-CNN, but not on par with more recent detectors (especially on small objects), also requiring extra-long training schedule and auxiliary decoding losses
  - DETR showed significant promise of generalizability, e.g., the same model easily applied to panoptic segmentation in a unified manner

“Pure Transformer”: Visual Transformer (ViT, ICLR’21)

GIF from https://github.com/lucidrains/vit-pytorch

Implementation

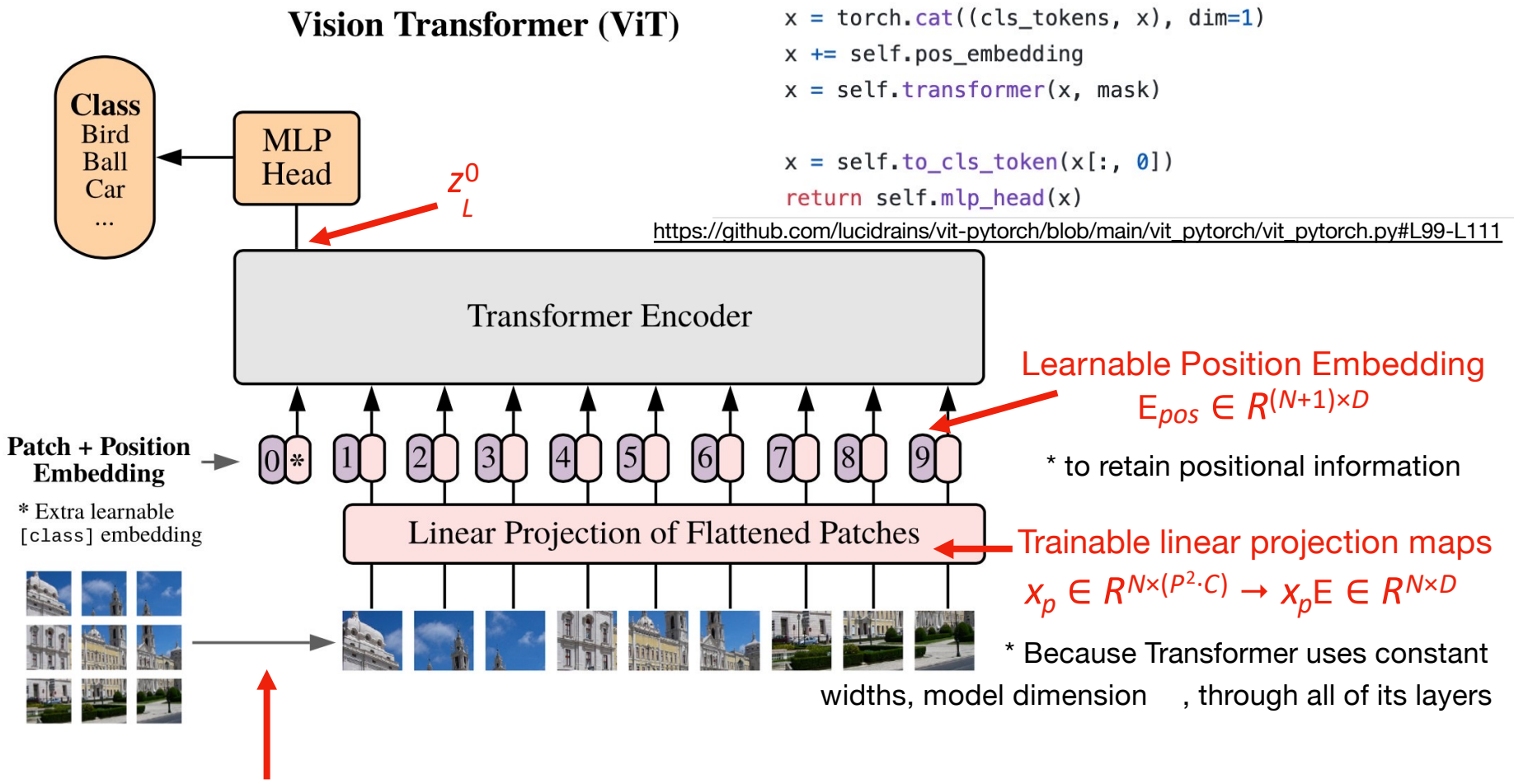
```
def forward(self, img, mask = None):
    p = self.patch_size

    x = rearrange(img, 'b c (h p1) (w p2) -> b (h w) (p1 p2
    x = self.patch_to_embedding(x)

    cls_tokens = self.cls_token.expand(img.shape[0], -1, -1)
    x = torch.cat((cls_tokens, x), dim=1)
    x += self.pos_embedding
    x = self.transformer(x, mask)

    x = self.to_cls_token(x[:, 0])
    return self.mlp_head(x)
```

https://github.com/lucidrains/vit-pytorch/blob/main/vit_pytorch/vit_pytorch.py#L99-L111



Implementation

```python
class Transformer(nn.Module):
    def __init__(self, dim, depth, heads, mlp_dim):
        super().__init__()
        self.layers = nn.ModuleList([])
        for _ in range(depth):
            self.layers.append(nn.ModuleList([
                Residual(PreNorm(dim, Attention(dim, heads = heads))),
                Residual(PreNorm(dim, FeedForward(dim, mlp_dim)))
            ]))
    def forward(self, x, mask = None):
        for attn, ff in self.layers:
            x = attn(x, mask = mask)
            x = ff(x)
        return x
```

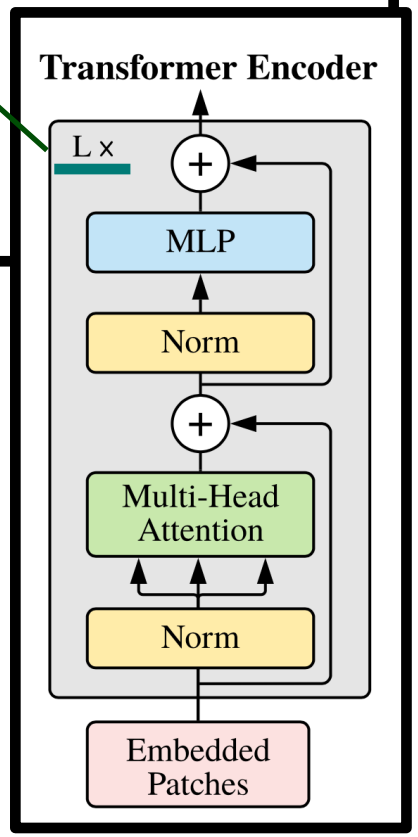
```python
def forward(self, img, mask = None):
    p = self.patch_size

    x = rearrange(img, 'b c (h p1) (w p2) -> b (h w) (p1 p2 c)', p1 = p, p2 = p)
    x = self.patch_to_embedding(x)

    cls_tokens = self.cls_token.expand(img.shape[0], -1, -1)
    x = torch.cat((cls_tokens, x), dim=1)
    x += self.pos_embedding
    x = self.transformer(x, mask)

    x = self.to_cls_token(x[:, 0])
    return self.mlp_head(x)
```

https://github.com/lucidrains/vit-pytorch/blob/main/vit_pytorch/vit_pytorch.py



Implementation

$z \in R^{N \times D}$: input sequence

$$[\mathbf{q}, \mathbf{k}, \mathbf{v}] = \mathbf{z}\mathbf{U}_{qkv} \qquad \mathbf{U}_{qkv} \in \mathbb{R}^{D \times 3D_h},$$

$$A = \operatorname{softmax}\left(\mathbf{q}\mathbf{k}^\top / \sqrt{D_h}\right) \qquad A \in \mathbb{R}^{N \times N},$$

$$\mathrm{SA}(\mathbf{z}) = A\mathbf{v}\,.$$

Attention weight A_{ij} : similarity btw q^i, k^j

$$\mathrm{MSA}(\mathbf{z}) = [\mathrm{SA}_1(z); \mathrm{SA}_2(z); \cdots; \mathrm{SA}_k(z)]\,\mathbf{U}_{msa} \qquad \mathbf{U}_{msa} \in \mathbb{R}^{k \cdot D_h \times D}$$

```python
class Attention(nn.Module):
    def __init__(self, dim, heads = 8):
        super().__init__()
        self.heads = heads
        self.scale = dim ** -0.5

        self.to_qkv = nn.Linear(dim, dim * 3, bias = False)
        self.to_out = nn.Linear(dim, dim)
    def forward(self, x, mask = None):
        b, n, _, h = *x.shape, self.heads
        qkv = self.to_qkv(x)
        q, k, v = rearrange(qkv, 'b n (qkv h d) -> qkv b h n d', qkv = 3, h = h)

        dots = torch.einsum('bhid,bhjd->bhij', q, k) * self.scale

        if mask is not None:
            mask = F.pad(mask.flatten(1), (1, 0), value = True)
            assert mask.shape[-1] == dots.shape[-1], 'mask has incorrect dimensions'
            mask = mask[:, None, :] * mask[:, :, None]
            dots.masked_fill_(~mask, float('-inf'))
            del mask

        attn = dots.softmax(dim=-1)

        out = torch.einsum('bhij,bhjd->bhid', attn, v)
        out = rearrange(out, 'b h n d -> b n (h d)')
        out =  self.to_out(out)
        return out
```

https://github.com/lucidrains/vit-pytorch/blob/main/vit_pytorch/vit_pytorch.py

Experiments

	Ours (ViT-H/14)	Ours (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	88.36	87.61 ± 0.03	87.54 ± 0.02	$88.4/\mathbf{88.5}^{*}$
ImageNet ReaL	**90.77**	90.24 ± 0.03	90.54	90.55
CIFAR-10	**99.50** ± 0.06	99.42 ± 0.03	99.37 ± 0.06	–
CIFAR-100	**94.55** ± 0.04	93.90 ± 0.05	93.51 ± 0.08	–
Oxford-IIIT Pets	**97.56** ± 0.03	97.32 ± 0.11	96.62 ± 0.23	–
Oxford Flowers-102	99.68 ± 0.02	**99.74** ± 0.00	99.63 ± 0.03	–
VTAB (19 tasks)	**77.16** ± 0.29	75.91 ± 0.18	76.29 ± 1.70	–
TPUv3-days	2.5k	0.68k	9.9k	12.3k

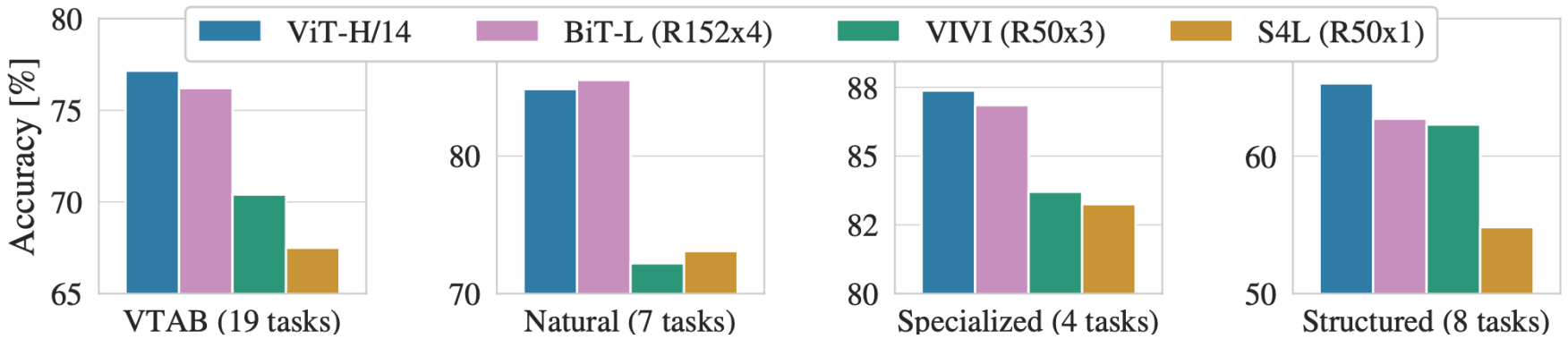


Figure 2: Breakdown of VTAB performance in *Natural*, *Specialized*, and *Structured* task groups.

Experiments

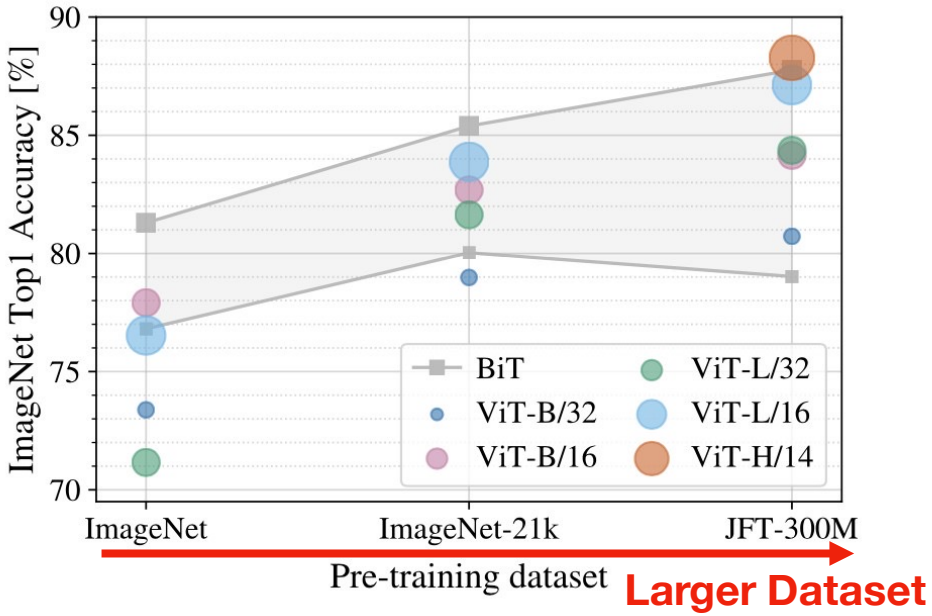


Figure 3: Transfer to ImageNet. While large ViT models perform worse than BiT ResNets (shaded area) when pre-trained on small datasets, they shine when pre-trained on larger datasets. Similarly, larger ViT variants overtake smaller ones as the dataset grows.

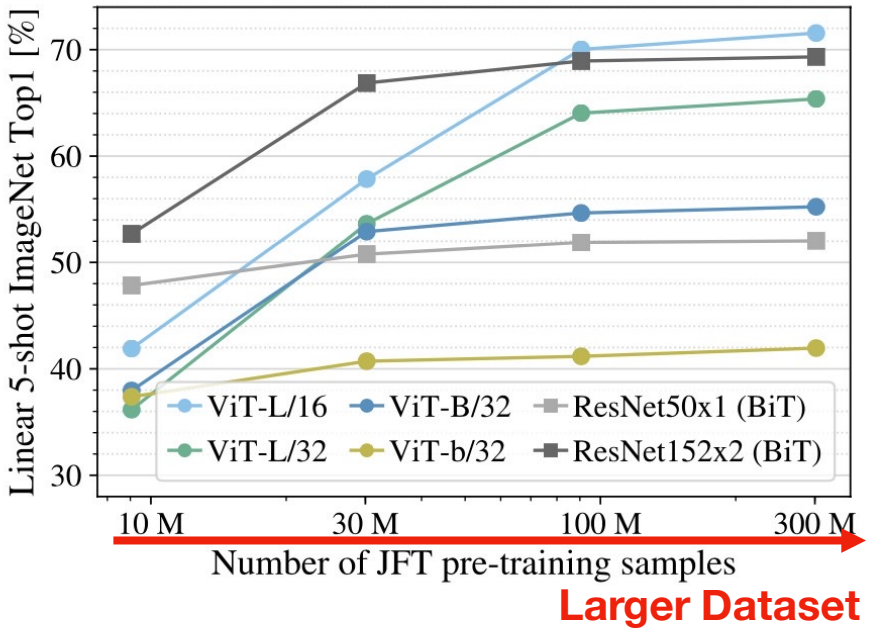
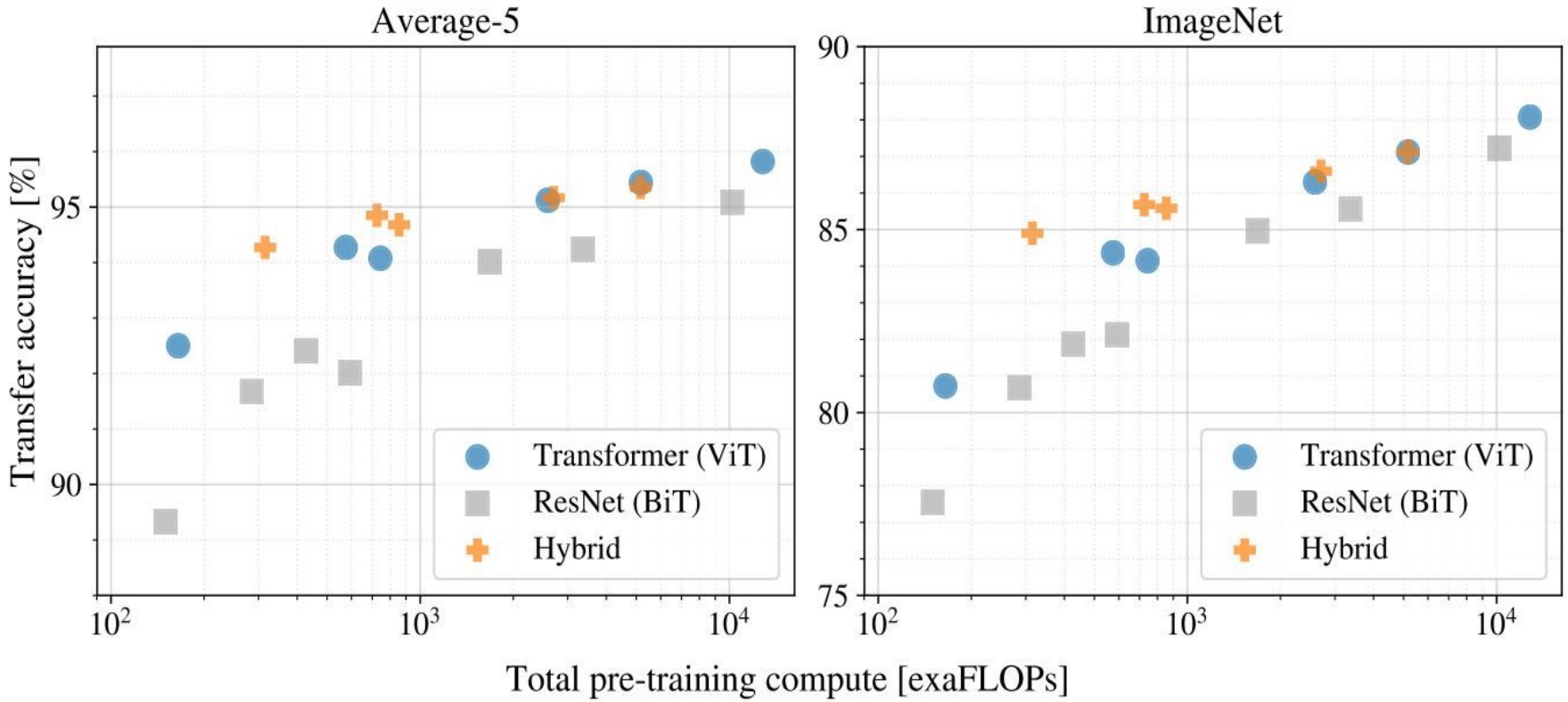


Figure 4: Linear few-shot evaluation on ImageNet versus pre-training size. ResNets perform better with smaller pre-training datasets but plateau sooner than ViT which performs better with larger pre-training. ViT-b is ViT-B with all hidden dimensions halved.

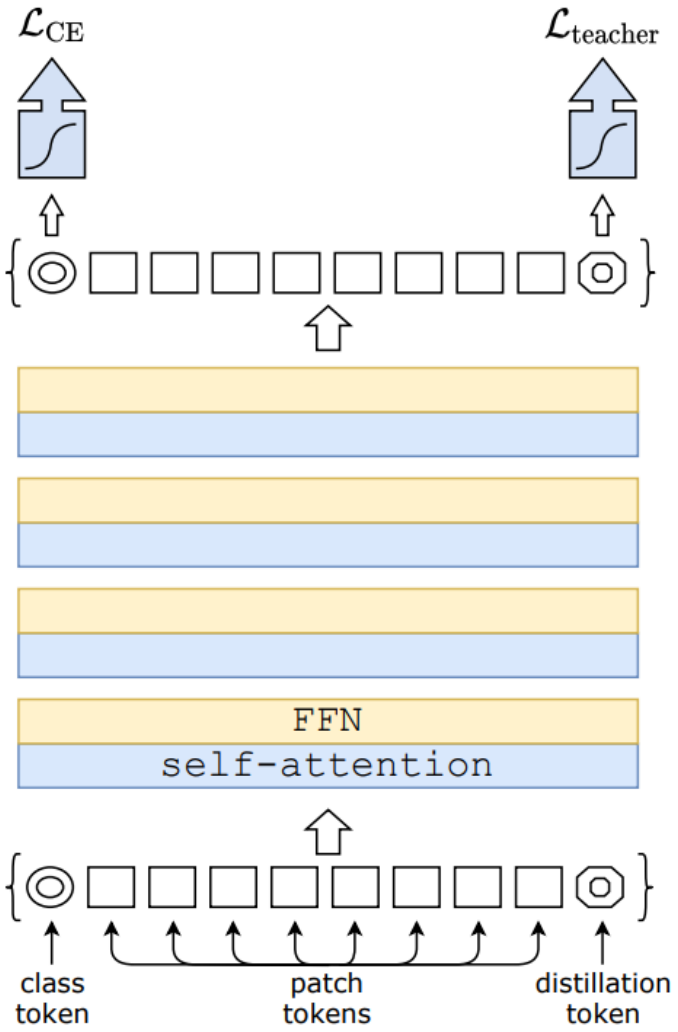
Experiments



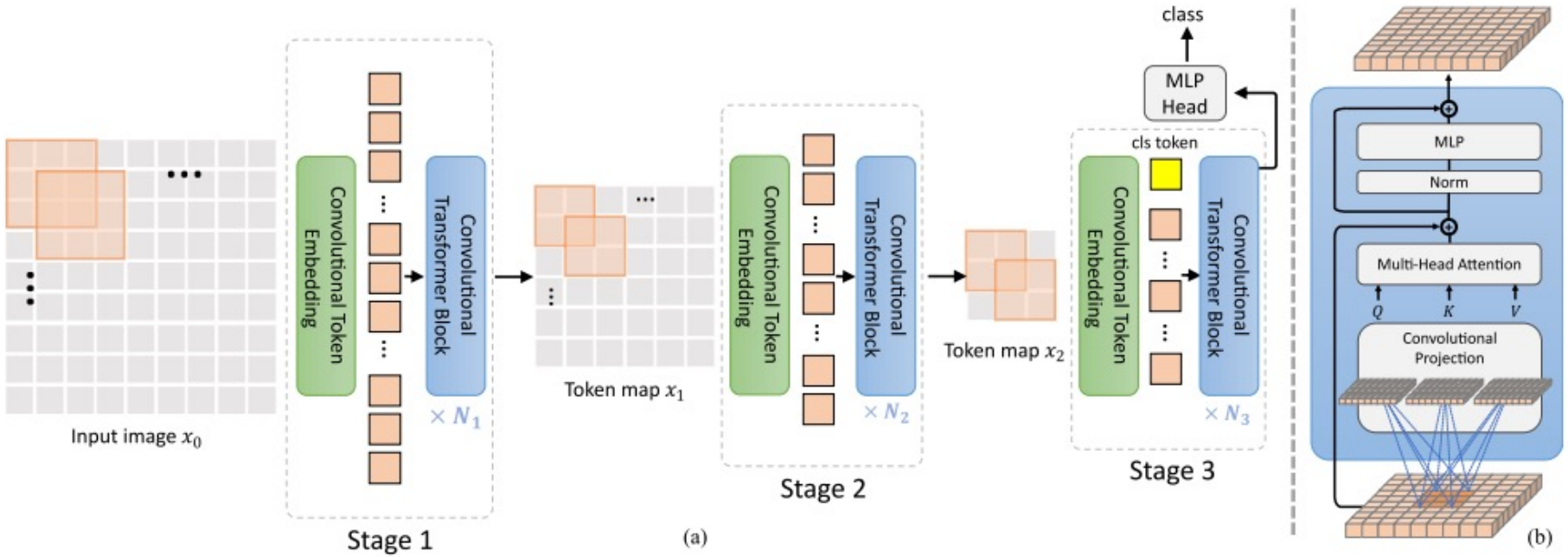
Performance versus cost for different architectures: Vision Transformers, ResNets, and hybrids. Vision Transformers generally outperform ResNets with the same computational budget. Hybrids improve upon pure Transformers for smaller model sizes, but the gap vanishes for larger models.

DeIT: Data-efficient Image Transformers

- The first competitive convolution-free transformer by training on Imagenet only
- Trained using a teacher-student strategy specific to transformers
  - It relies on a distillation token ensuring that the student learns from the teacher through attention.
- When using CNN as teacher, the distilled model outperforms its teacher in terms of the trade-off between accuracy and throughput

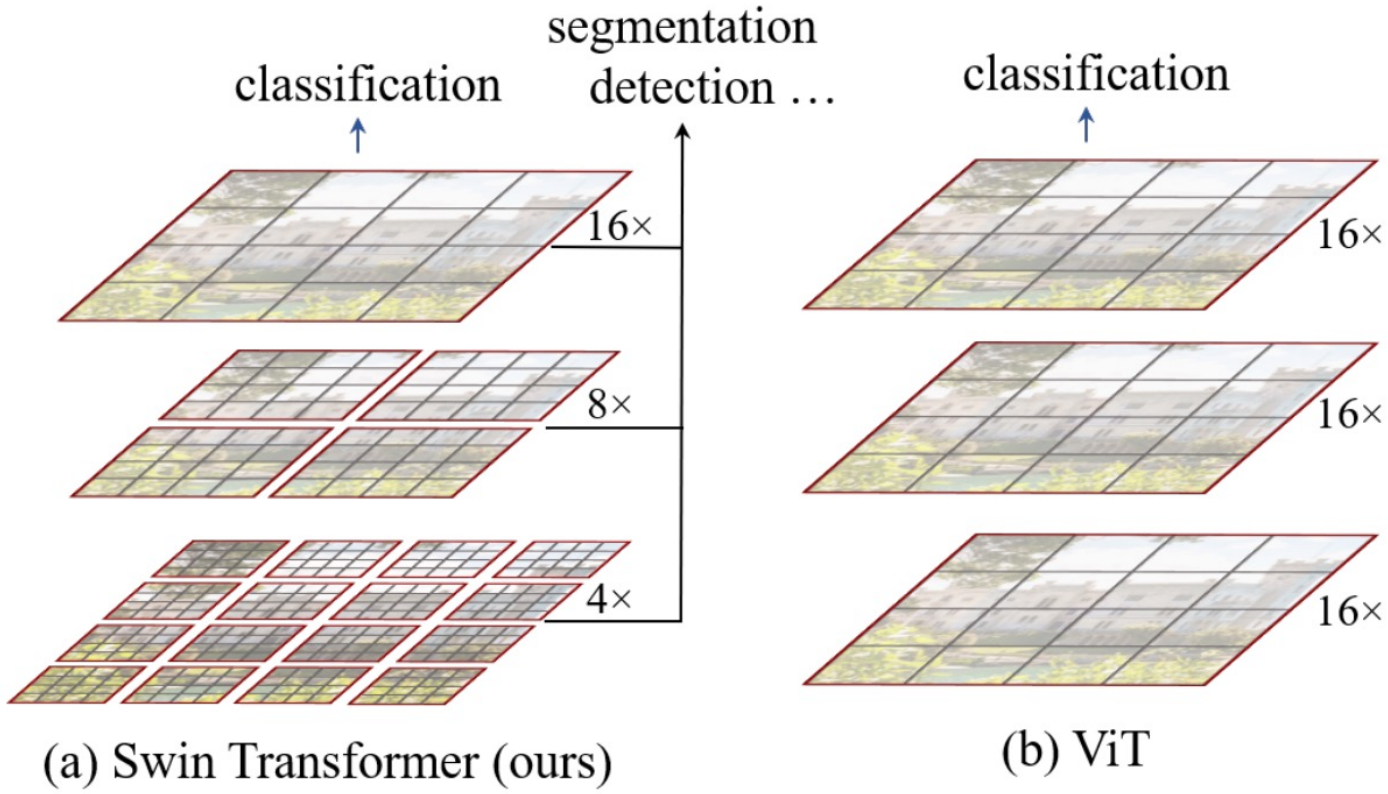


CvT: Convolutions into Vision Transformers



- Each stage starts with a convolutional token embedding that performs an overlapping convolution operation on a 2D-reshaped token map
- The linear projection prior to every self-attention block is replaced with a depth-wise separable convolution as the projection

Swin Transformer (ICCV'21 best paper)



- Swin: hierarchical feature maps by merging image patches
  - linear computation complexity to input image size due to computation of self-attention only within each local window (using **S**hifted **win**dows)

Swin Transformer: Pipeline Overview

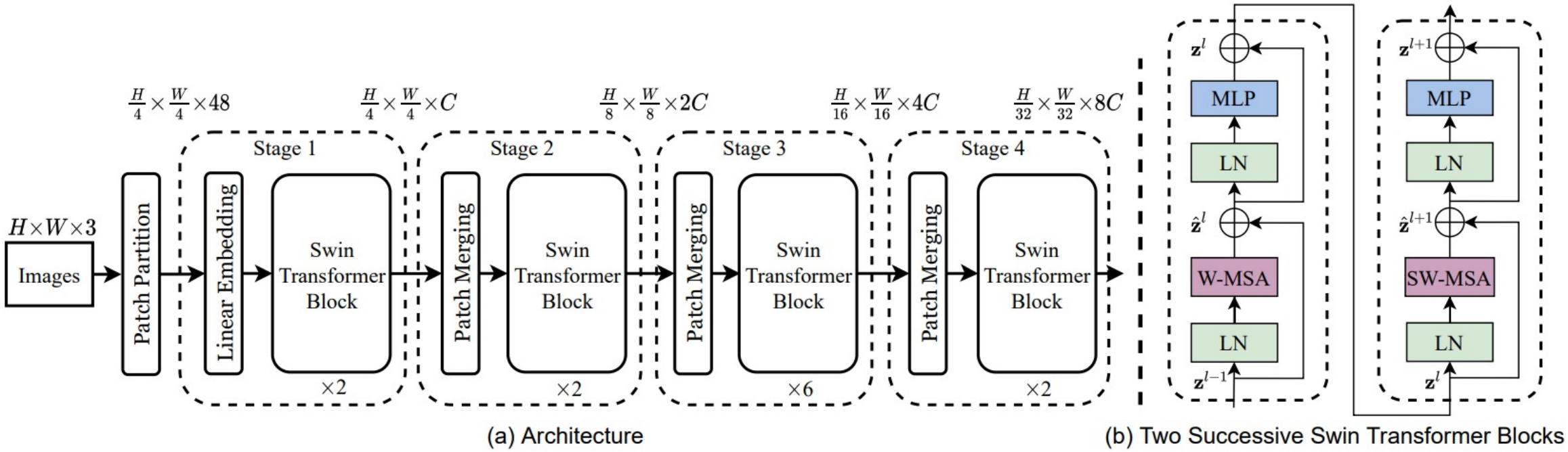
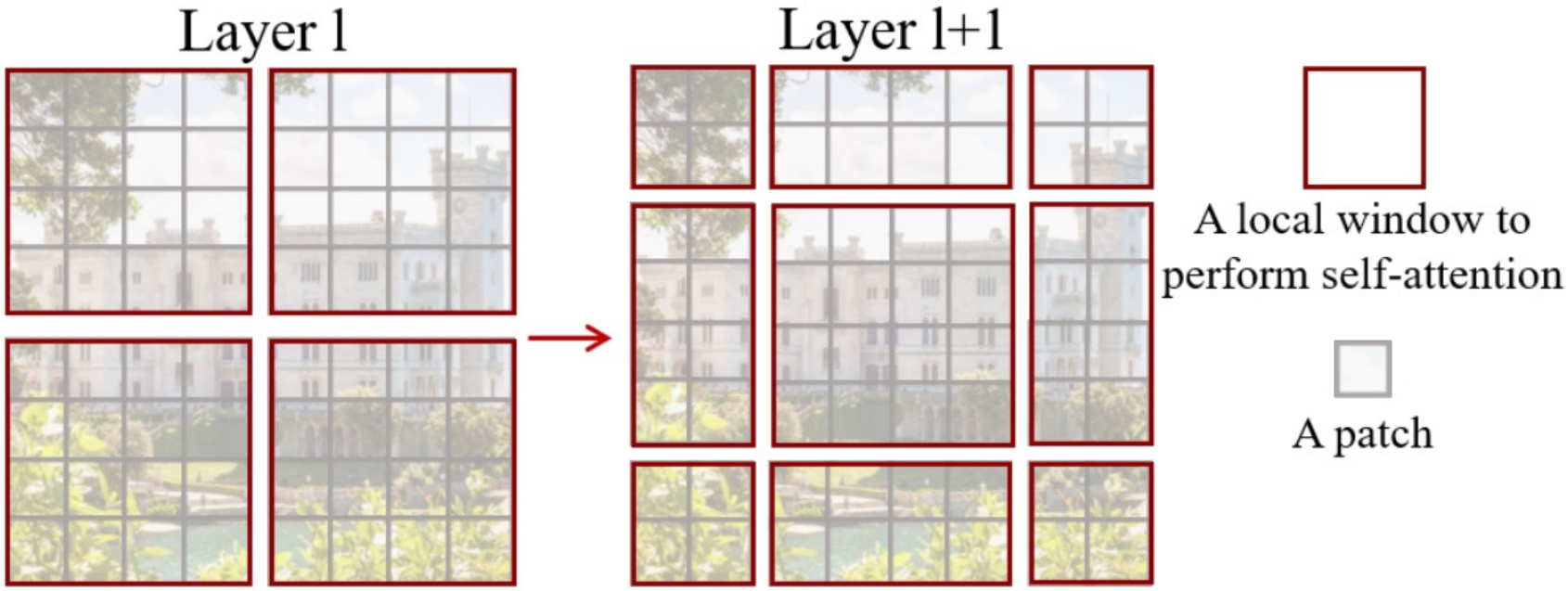
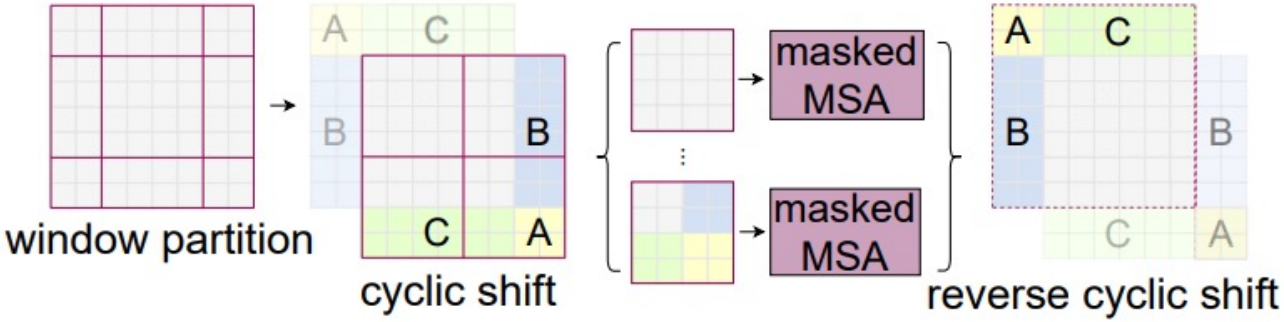


Figure 3. (a) The architecture of a Swin Transformer (Swin-T); (b) two successive Swin Transformer Blocks (notation presented with Eq. (3)). W-MSA and SW-MSA are multi-head self attention modules with regular and shifted windowing configurations, respectively.

Swin Transformer: Shifted Window





TimeSformer: ViT for Video

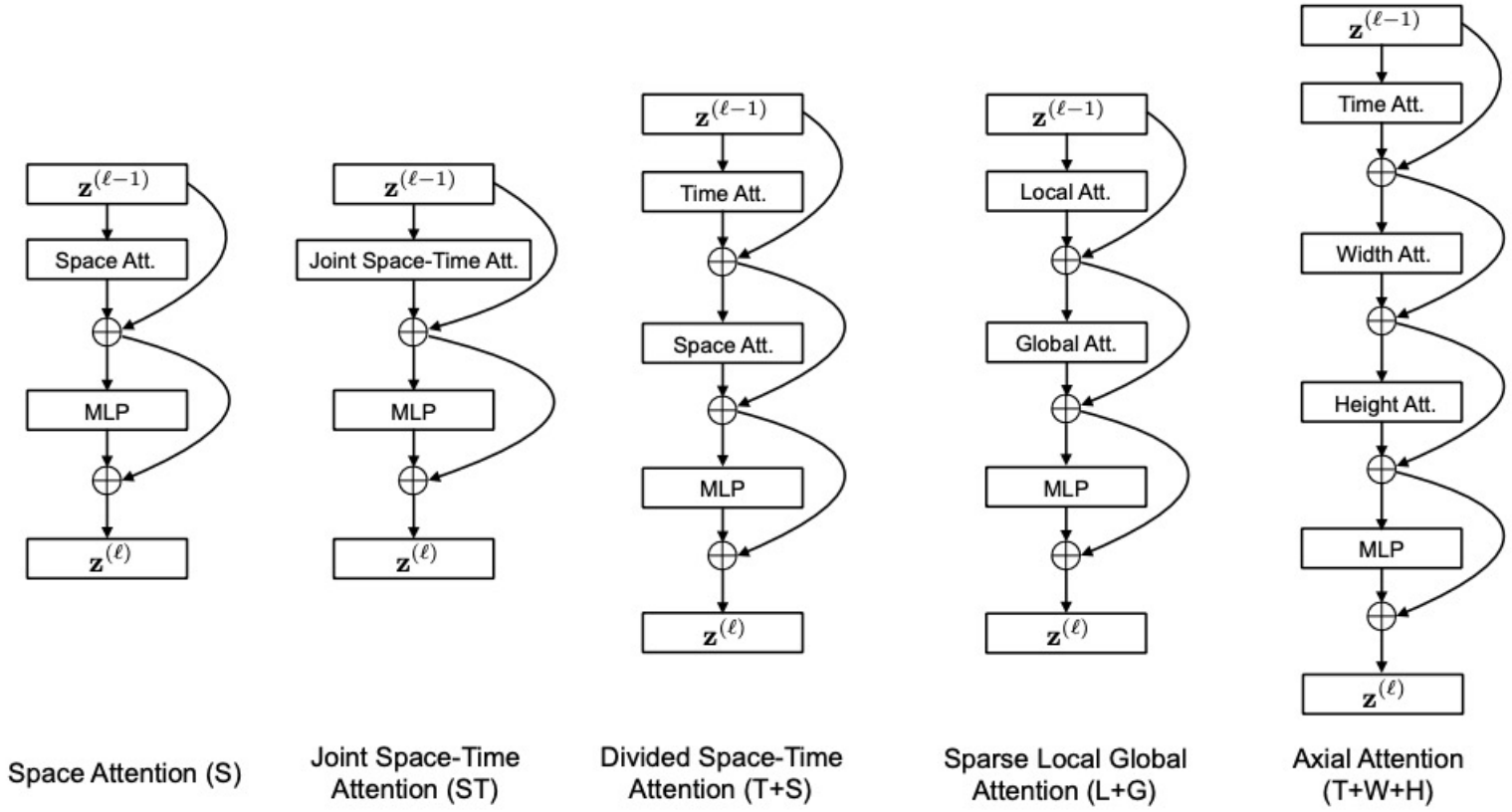


Figure 1. The video self-attention blocks that we investigate in this work. Each attention layer implements self-attention (Vaswani et al., 2017b) on a specified spatiotemporal neighborhood of frame-level patches (see Figure 2 for a visualization of the neighborhoods). We use residual connections to aggregate information from different attention layers within each block. A 1-hidden-layer MLP is applied at the end of each block. The final model is constructed by repeatedly stacking these blocks on top of each other.

TimeSformer: ViT for Video

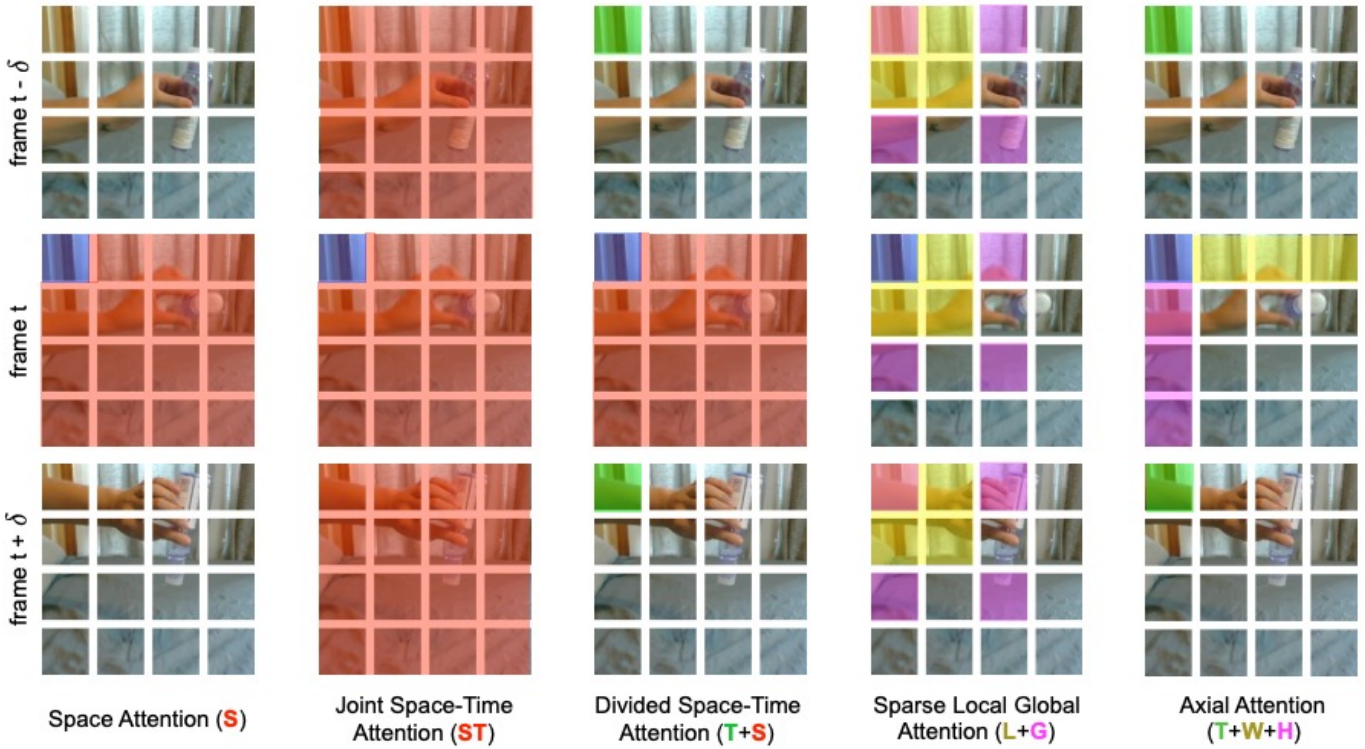


Figure 2. Visualization of the five space-time self-attention schemes studied in this work. Each video clip is viewed as a sequence of frame-level patches with a size of 16×16 pixels. For illustration, we denote in blue the query patch and show in non-blue colors its self-attention space-time neighborhood under each scheme. Patches without color are not used for the self-attention computation of the blue patch. Multiple colors within a scheme denote attentions separately applied along different dimensions (e.g., space and time for (T+S)) or over different neighborhoods (e.g., for (L+G)). Note that self-attention is computed for every single patch in the video clip, i.e., every patch serves as a query. We also note that although the attention pattern is shown for only two adjacent frames, it extends in the same fashion to all frames of the clip.

TimeSformer: ViT for Video

Attention	Params	K400	SSv2
Space	85.9M	76.9	36.6
Joint Space-Time	85.9M	77.4	58.5
Divided Space-Time	121.4M	**78.0**	**59.5**
Sparse Local Global	121.4M	75.9	56.3
Axial	156.8M	73.5	56.2

Table 1. Video-level accuracy for different space-time attention schemes in TimeSformer. We evaluate the models on the validation sets of Kinetics-400 (K400), and Something-Something-V2 (SSv2). We observe that divided space-time attention achieves the best results on both datasets.

Model	Pretrain	K400 Training Time (hours)	K400 Acc.	Inference TFLOPs	Params
I3D 8x8 R50	ImageNet-1K	444	71.0	1.11	28.0M
I3D 8x8 R50	ImageNet-1K	1440	73.4	1.11	28.0M
SlowFast R50	ImageNet-1K	448	70.0	1.97	34.6M
SlowFast R50	ImageNet-1K	3840	75.6	1.97	34.6M
SlowFast R50	N/A	6336	76.4	1.97	34.6M
TimeSformer	ImageNet-1K	**416**	75.8	**0.59**	121.4M
TimeSformer	ImageNet-21K	**416**	**78.0**	**0.59**	121.4M

Table 2. Comparing TimeSformer to SlowFast and I3D. We observe that TimeSformer has lower inference cost despite having a larger number of parameters. Furthermore, the cost of training TimeSformer on video data is much lower compared to SlowFast and I3D, even when all models are pretrained on ImageNet-1K.

Method	Top-1	Top-5	TFLOPs
R(2+1)D (Tran et al., 2018)	72.0	90.0	17.5
bLVNet (Fan et al., 2019)	73.5	91.2	0.84
TSM (Lin et al., 2019)	74.7	N/A	N/A
S3D-G (Xie et al., 2018)	74.7	93.4	N/A
Oct-I3D+NL (Chen et al., 2019)	75.7	N/A	0.84
D3D (Stroud et al., 2020)	75.9	N/A	N/A
I3D+NL (Wang et al., 2018b)	77.7	93.3	10.8
ip-CSN-152 (Tran et al., 2019)	77.8	92.8	3.2
CorrNet (Wang et al., 2020a)	79.2	N/A	6.7
LGD-3D-101 (Qiu et al., 2019)	79.4	94.4	N/A
SlowFast (Feichtenhofer et al., 2019b)	79.8	93.9	7.0
X3D-XXL (Feichtenhofer, 2020)	80.4	94.6	5.8
TimeSformer	78.0	93.7	**0.59**
TimeSformer-HR	79.7	94.4	5.11
TimeSformer-L	**80.7**	**94.7**	7.14

Table 5. Video-level accuracy on Kinetics-400.

DINO: Self-Supervised Learning with ViTs

Source: https://ai.facebook.com/blog/dino-paws-computer-vision-with-self-supervised-transformers-and-10x-more-efficient-training/

DINO: Self-Supervised Learning with ViTs

DINO: Self-Supervised Learning with ViTs

Supervised

DINO

Multi-Modality: Video-Audio-Text Transformer (VATT)

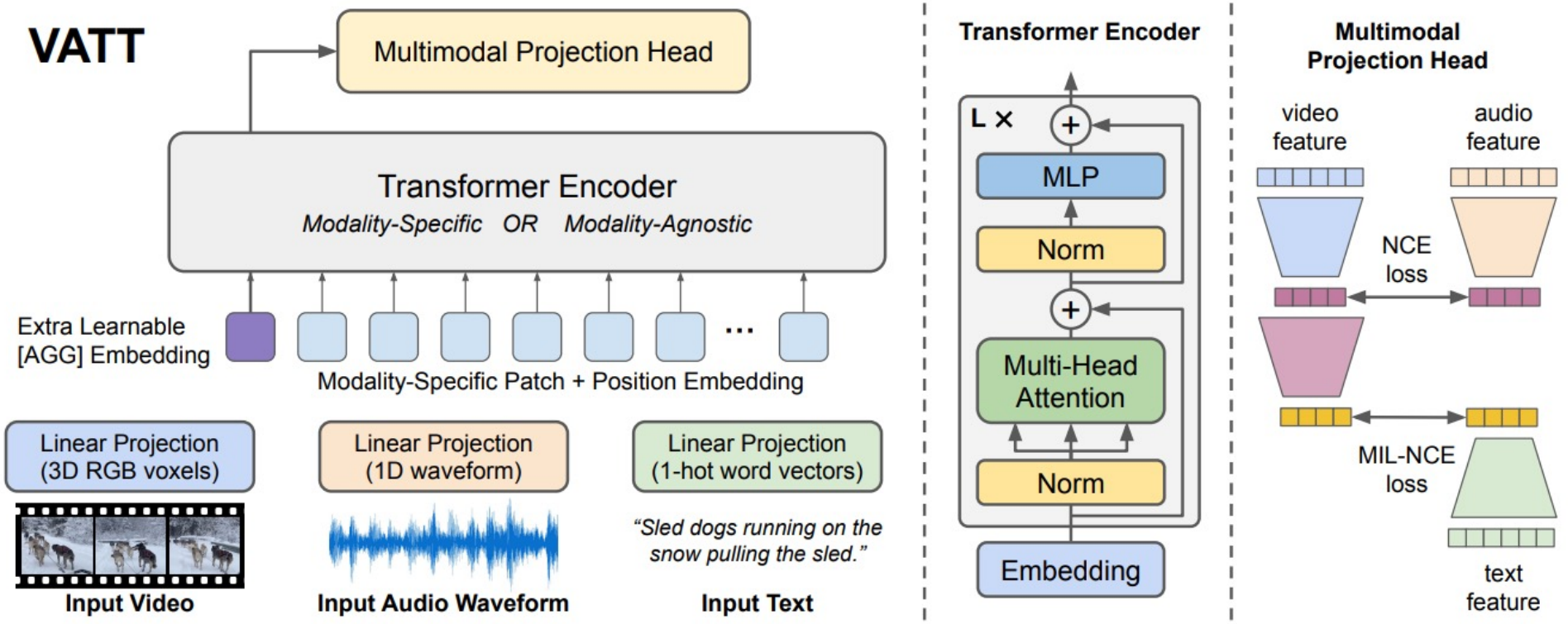


Figure 1: **Overview of the VATT architecture and the self-supervised, multimodal learning strategy**. VATT linearly projects each modality into a feature vector and feeds it into a Transformer encoder. We define a semantically hierarchical common space to account for the granularity of different modalities and employ the Noise Contrastive Estimation (NCE) to train the model.

Ongoing Debate: ViTs Should Go More Complicated or Less?

- Adding “convolution-like” inductive bias and structures
  - Injecting convolution layers, pyramid structure, dense connections, sliding windows, multi-sized views or attention windows …
- … Or just, keep it simple and “universal”?
  - **Example:** W. Chen et. al., “A Simple Single-Scale Vision Transformer for Object Detection and Instance Segmentation”, ECCV 2022
  - Someone goes even further: MLP-Mixer, Conv-Mixer, Perceiver-IO…

Look Back: ConvNets

- Inductive biases
- Translation equivariance
- Shared computations
- Hierarchical feature maps

- Typical build: ”going deeper with small convolutions”
- Pros versus Cons?

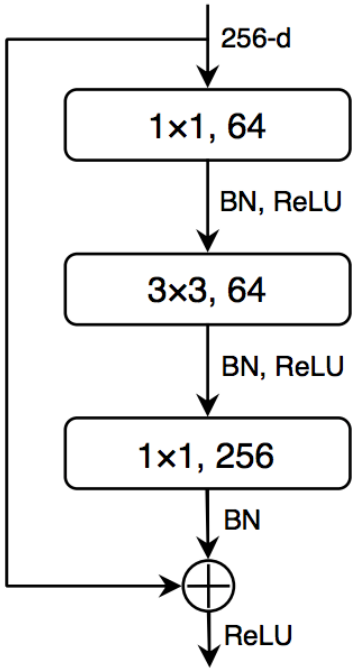
Look Back: Vision Transformers

- Plain transformers outperform ResNets by a significant margin
  - mostly on image classification, only recently on detection/segmentation
- No hierarchical feature maps
- Quadratic complexity with respect to the input size

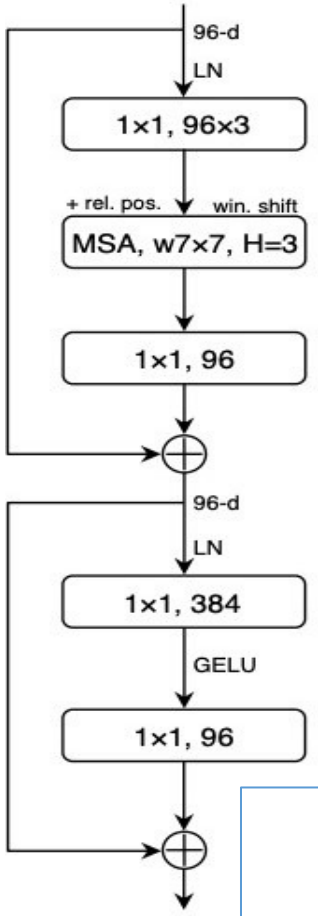
No silver bullets! What is in-between ConvNets and ViTs?

ConvNeXt – 7x7 (Liu et al. 2022)

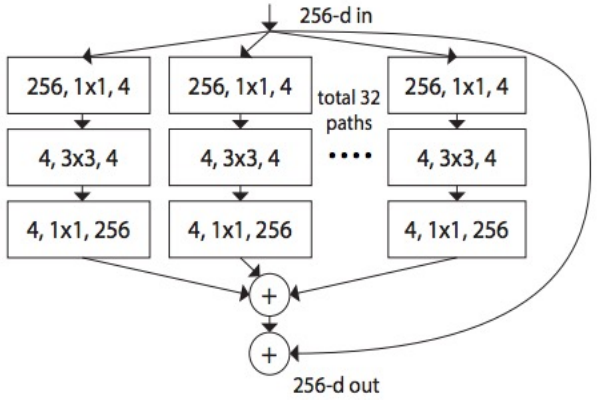
ConvNet



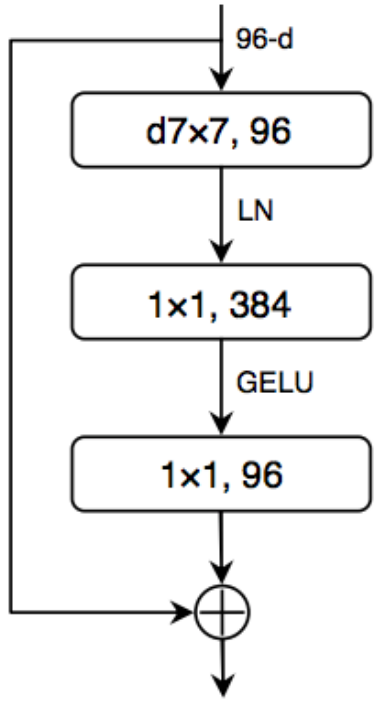
Swin Transformer



ResNeXt



ConvNeXt!



A ConvNet for the 2020s

Zhuang Liu[1,2*] Hanzi Mao[1] Chao-Yuan Wu[1] Christoph Feichtenhofer[1] Trevor Darrell[2] Saining Xie[1†]

[1]Facebook AI Research (FAIR) [2]UC Berkeley

ConvNeXt – 7x7

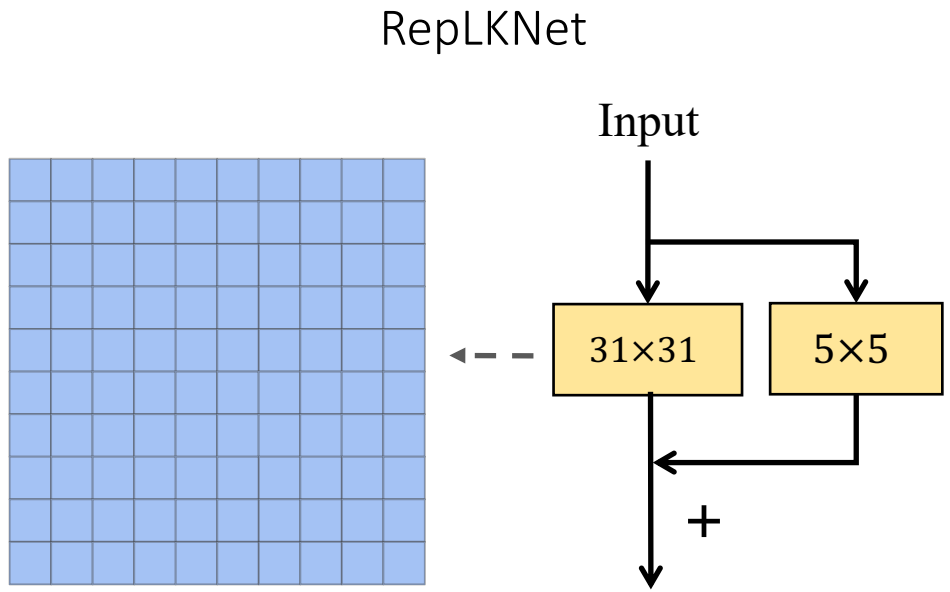
- ConvNeXts compete favourably with Transformers on image classification.
- ConvNeXts outperforms Swin Transformer on general computer vision tasks such as object detection and semantic segmentation.

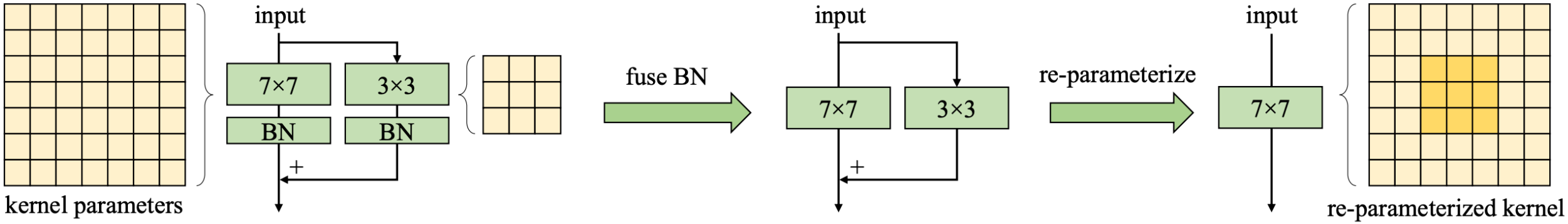
RepLKNet – 31x31

- Large Kernels + Structural Re-parameterization

⇩

- Achieving comparable or superior results than Swin on ImageNet + a few downstream tasks.

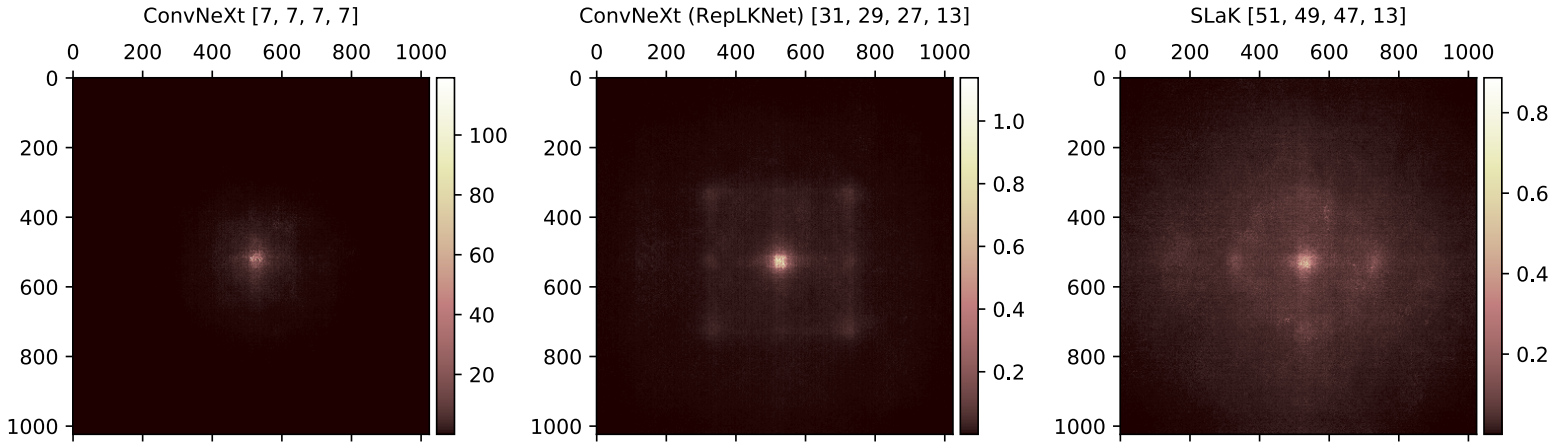




Sparse Large Kernel Network (SLaK)

Starting from ConvNeXt …

- Increase the kernel size of stages to [51, 49, 47, 13]
- Construct sparse decomposed kernels (sparsity=0.4, N=5)
- Use sparse groups, expand model width to 1.3x



VITA

The University of Texas at Austin
Electrical and Computer Engineering
Cockrell School of Engineering